IL BASIC E LA GESTIONE DEI FILE

Metodi Pratici

1° Volume

EDIZIONE ITALIANA

GRUPPO EDITORIALE JACKSON

Jacques Boisgontier

IL BASIC E LA GESTIONE DEI FILE

Metodi Pratici

1° Volume

di
Jacques Boisgontier

GRUPPO
EDITORIALE
JACKSON
Via Rosellini, 12
20124 Milano

AVVERTENZE

Quest'opera è destinata a coloro che desiderano realizzare dei programmi, da utilizzare su Personal Computers, che prevedano *l'uso di files residenti su disco.*

Dato che esistono molteplici versioni del linguaggio Basic e dei sistemi di gestione per i dischi e per i floppy-disk, dovendo scegliere abbiamo optato per quella che, secondo noi, dava maggiori garanzie di generalità.

Si tratta della *versione Basic 5., CP/M compatibile*, realizzata dalla Microsoft. Questa versione del Basic è stata realizzata in particolare per i *sistemi basati sul microprocessore Z80* ma con *qualche lieve modifica* può essere utilizzata anche su sistemi basati sul microprocessore 6800 (ad esempio il sistema X1 della S.O.E).

Al momento attuale questa versione del Basic è utilizzata su molti Personal Computers, tra i quali vogliamo citare i più noti: TRS80 (con qualche restrizione che verrà segnalata di volta in volta); Zenith (Heatkit) Z8,Z89; LX500 con S.E.D. SX/80; Occitane X1; North star; Cromenco; Altos; Sanco; Vector; Micral 80; PCC 2000; Kontron PS1 80; ISTC 5000 option CP/M; DTC Microfile; CB 7900; Industrial Micro System; ...

Il Gruppo Editoriale Jackson ringrazia per il prezioso lavoro svolto nella stesura dell'edizione italiana la signora Francesca di Fiore, e l'Ing. Roberto Pancaldi.

Prima edizione novembre 1982
Stampato in Italia da:
S.p.A. Alberto Matarelli — Milano — Stabilimento Grafico

Fotocomposizione:
CorpoNove s.n.c. — Bergamo — via Borfuro 14/c — Tel. 22.33.63-22.33.65

SOMMARIO

PRESENTAZIONE

Quest'opera si rivolge in modo particolare ai lettori che conoscono già le tecniche generali di programmazione in Basic. *Per coloro che invece non hanno mai avuto occasione di utilizzare questo tipo di linguaggio, vengono richiamate le caratteristiche fondamentali nell'Appendice 1.*

Riteniamo comunque interessante per tutti una lettura iniziale, seppur rapida, del contenuto di questa appendice.

Il presente volume è diviso logicamente in tre parti distinte:

La prima, dedicata alla descrizione delle particolarità della versione Basic Microsoft 5., è corredata da numerosi esempi pratici. Si consiglia a questo proposito al lettore di confrontare questa versione del Basic con quella descritta sul manuale d'uso del proprio Personal Computer, al fine di mettere in luce eventuali differenze.

La seconda è dedicata alla descrizione delle istruzioni necessarie per una corretta gestione dei files su disco, sia ad accesso diretto che sequenziale. Anche in questo caso sono stati prodotti numerosi esempi illustrativi.

La terza parte è interamente dedicata all'esposizione dei metodi pratici per l'uso dei files ad accesso diretto e dei data base.

L'opera è poi completata da due appendici tecniche sui messaggi d'errore e sui codici ASCII. Vi è inoltre un elenco alfabetico dei termini tecnici utilizzati con i relativi riferimenti.

Per evitare al massimo gli errori di trascrizione, i tabulati relativi ai programmi sono stati direttamente riprodotti dagli originali ricavati dall'elaboratore.

NOTA: Nei programmi, a volte, sono state tralasciate alcune tra le routine più semplici, lasciando al lettore, quale esercizio, lo sviluppo delle stesse.

CAPITOLO 1

IL BASIC MICROSOFT 5

BATTITURA DI UN PROGRAMMA CODIFICATO

Ciascuna riga di programma deve essere numerata. I numeri di riga ammessi vanno da 1 a 65529. Le varie righe possono essere trasmesse al calcolatore, purchè opportunamente numerate, in un ordine qualsiasi.

Il numero massimo di caratteri ammesso per ogni riga è 255.

Il carattere "." permette di far riferimento, in un comando di consolle, alla riga corrente.

"?" equivale a PRINT.

Il tasto LINE-FEED permette di scrivere un'unica riga logica di programma utilizzando fisicamente più righe sul video.

Tutto ciò che in una riga di programma segue la parola REM o è racchiuso tra apici viene considerato come commento.

Si possono scrivere più istruzioni su una stessa riga: in tal caso le singole istruzioni devono essere separate con il carattere":" (due punti).

Se, durante l'esecuzione del programma, l'interprete Basic incontra un errore di sintassi, trasmette all'utente un messaggio indicante il tipo di errore e la riga in cui tale errore è stato riscontrato, sospende l'esecuzione del programma e passa il controllo all'Editor per la correzione:

```
          10 BUONGIORNO
     RUN
          SYNTAX ERROR IN 10
       10
```

Il programmatore può in questo caso controllare il contenuto della riga 10 ed eventualmente apportarvi delle modifiche. Se però prima di effettuare le modifiche si vuole controllare il valore assunto da alcune variabili a quel punto del programma, occorre rilasciare l'EDIT mediante il comando Control-Q dato che anche una sola modifica

apportata al programma comporterebbe una riinizializzazione a zero di tutte le variabili utilizzate dal programma stesso.

Per rilasciare l'Edit con la riinizializzazione automatica di tutte le variabili è sufficiente invece premere il tasto RETURN (su alcune tastiere CR).

Caratteri di controllo

Control A: Permette di utilizzare l'EDIT sulla riga corrente
Control C: Interrompe l'esecuzione del programma in corso e passa il controllo al BASIC
Control G: Attiva il segnale sonoro
Control H: Cancella l'ultimo carattere battuto
Control I: Predispone la tabulazione (di 8 in 8)
Control O: Inibisce l'operazione di stampa (o di visualizzazione) senza però sospendere l'esecuzione del programma.
Per riattivare la stampa basta dare un nuovo Control O.
Control R: Visualizza la riga corrente
Control S: Sospende l'esecuzione del programma
Control Q: Riattiva l'esecuzione sospesa
Control U: Annulla la riga corrente

Editor microsoft 5.

Una riga di programma può essere interamente riscritta.

Se durante la battitura di un'istruzione, ci si accorge di aver battuto un carattere errato si può cancellare l'ultimo carattere battuto mediante uno dei due comandi:

Control H o RUBOUT

Se invece si vuole cancellare l'intera riga basta utilizzare il comando:

Control U

Si possono inoltre modificare delle righe di programma già memorizzate in precedenza. Per far ciò occorre applicare l'EDIT alla riga che si desidera modificare, utilizzando il comando

EDIT n° di riga

Esempio:
EDIT 100

Funzioni dell'EDIT

Posizionamento del cursore	ESPACE (barra spaziatrice):	Battendo questo tasto il cursore si sposta ogni volta di uno spazio verso destra. Mediante la forma xESPACE si ottiene direttamente lo spostamento del cursore di x spazi verso destra.
	RUBOUT:	Con questo comando si ottiene lo spostamento del cursore di uno spazio verso sinistra. Utilizzando la forma xRUBOUT si ottiene direttamente lo spostamento del cursore di x posti a sinistra.
Inserimento di caratteri	I (insert):	Per inserire in una riga uno o più caratteri basta, in EDIT, battere il carattere I seguito dalla striga che si vuole inserire a partire dalla posizione corrente del cursore. L'operazione di inserimento si conclude battendo il tasto ESCAPE. Se non si devono effettuare altre modifiche si può rilasciare l'EDIT mediante un CR, altrimenti si possono effettuare altre modifiche utilizzando gli opportuni comandi.
Aggiunta di caratteri alla fine di una riga	X:	Mediante la battitura di questo carattere si posiziona il cursore alla fine della riga in esame. A questo punto basta aggiungere la parte di testo voluta.
Cancellazione di caratteri	D: (delete)	Battendo il carattere D si cancella il carattere su cui è posizionato il cursore. Mediante il comando xD si cancellano x caratteri a partire da quello individuato dal cursore.
	H (hack):	Cancella tutti i caratteri alla destra del cursore e permette poi l'inserimento di un nuovo testo.
Ricerca di un carattere	S (search):	'S *carattere cercato*' posiziona il cursore sul primo carattere uguale a quello indicato presente nella riga. 'i S *carattere cercato*' posiziona il cur-

		sore sull'i-esima occorrenza del carattere specificato.
	K (search and kill):	'i K *carattere cercato*' posiziona il cursore sull'i-esima occorrenza del carattere specificato e cancella tutti i caratteri che precedono la posizione raggiunta.
Sostituzione di caratteri	C (change):	'C *carattere*' cancella il carattere successivo a quello su cui è posizionato il cursore e lo sostituisce con *carattere*. 'k C *stringa*' cancella K caratteri a partire da quello successivo alla posizione del cursore e li sostituisce con quelli contenuti in '*stringa*'. Dopo l'operazione di sostituzione il controllo è restituito all'EDIT.
Rilascio dell'EDIT	RETURN (CR):	Con il RETURN si ottiene il rilascio dell'EDIT, la memorizzazione delle correzioni e la visualizzazione dell'intera riga corretta.
	E (exit):	Ha la stessa funzione del RETURN ma non visualizza la riga corretta.
	Q (cancel and quite):	Viene rilasciato l'EDIT ma non vengono memorizzate le modifiche apportate.
	L (list):	Si mantiene in fase di EDIT visualizzando l'intera riga e riposizionando il cursore sul primo carattere.
	A (annulle):	Annulla tutte le modifiche effettuate e riposiziona il cursore all'inizio della riga.

COMANDI DEL BASIC MICROSOFT 5.

Questi comandi possono essere utilizzati direttamente da consolle, dopo la visualizzazione di un OK da parte del sistema, oppure all'interno di un programma.

AUTO n° *riga, incremento*: genera automaticamente un nuovo numero di riga ogni volta che viene battuto un RETURN. Si può interrompere la generazione automatica mediante un Control C.

AUTO 100,5 genera i numeri di riga a partire da 100 con un incremento di 5 ogni volta.

AUTO 100 genera la numerazione a partire da 100 con un incremento di 10

CLEAR, *indirizzo max, spazio per le stringhe:* azzera tutte le variabili, fissa l'indirizzo più alto utilizzabile dal Basic e riserva l'area destinata ai dati alfanumerici. Se *'spazio per stringhe'* non viene specificato viene automaticamente riservata un'area pari o a 1000 bytes o a 1/8 della memoria disponibile (viene considerato il valore minore).

CLEAR, 3000 azzera tutte le variabili e riserva 3000 bytes per le stringhe.
CLEAR azzera tutte le variabili e riserva lo spazio standard per le stringhe.

CONT: fa riprendere l'esecuzione di un programma interrotto con uno STOP o un END o un Control C. Il comando CONT non viene accettato se dopo la sospensione si sono apportate delle modifiche al programma (in questo caso si deve ricorrere al comando GOTO xx).

DELETE n° *riga iniziale — riga finale:* sopprime tutte le righe di programma i cui numeri identificatori sono compresi tra quelli indicati. *'riga iniziale'* e *'riga finale'* devono corrispondere a righe realmente esistenti.

DELETE 100—200 cancella tutte le righe numerate tra 100 e 200.

FILES *"U:x.x":* fornisce l'elenco dei files presenti sull'unità a disco specificata (U=A, B, C ...)

FILES "A: x. BAS" fornisce la lista dei files di sistema (BASIC) presenti sul disco A

KILL *"U: nome del file"*: cancella il file indicato presente sull'unità disco specificata.

KILL "A: PIPPO" cancella il file PIPPO dal disco A

LIST *n° linea iniziale – n° linea finale*: visualizza tutte le righe del programma aventi i numeri compresi tra quelli indicati

LIST	visualizza tutte le istruzioni del programma
LIST 10	visualizza solo la riga 10
LIST 500–	visualizza la parte di programma compresa tra la riga 500 e la fine
LIST–400	visualizza tutte le righe di programma fino alla 400.

LLIST Stampa il programma completo utilizzando la stampante.

LOAD *"U: nome del file"*, R: carica in memoria centrale il file indicato e ne lancia l'esecuzione (R)

LOAD "PIPPO"	carica in memoria il programma PIPPO residente sul disco A
LOAD "B: PIPPO"	carica in memoria il programma PIPPO residente sul disco B
LOAD "PIPPO", R	carica in memoria il programma PIPPO residente sul disco A e lo esegue.

NAME *"vecchio nome" AS "nuovo nome"*: modifica il nome di un file
NAME "PIPPO" AS "PLUTO"

NEW cancella il programma presente in memoria.

MERGE *"U: nome del file"*: concatena il programma già presente in memoria con il programma specificato (purchè memorizzato in forma ASCII). Se alcune righe del programma richiamato hanno lo stesso numero di identificazione di quelle presenti nel programma residente, esse le sostituiscono in fase di concatenazione.

RENUM *nuovo numero di riga, prima riga da rinumerare, incremento*:
rinumera le righe del programma.
- *il 'nuovo numero di riga'* rappresenta il numero che deve assumere la prima riga da rinumerare (se non specificato viene assunto uguale a 10).
- *la 'prima riga da rinumerare'* indica da che punto del programma deve iniziare la rinumerazione. Se questa opzione non è specificata viene rinumerato tutto il programma.

— *'incremento'* indica il valore dell'incremento che deve essere usato nella rinumerazione. Se questa opzione non è utilizzata l'incremento viene assunto uguale a 10.

RENUM	rinumera tutte le righe del programma assegnando alla prima riga il numero 10 e utilizzando un incremento pari a 10
RENUM 1000,,10	rinumera tutte le righe del programma assegnando alla prima riga il numero 1000 e utilizzando un incremento pari a 10
RENUM 1000,900,10	rinumera il programma della riga 900 in poi, assegnando alla riga 900 il nuovo numero 1000 ed utilizzando un incremento di 10 per rinumerare le successive.

RUN *n° di riga*: lancia l'esecuzione del programma presente in memoria a partire dall'istruzione specificata. Tutte le variabili vengono inizializzate a zero.

RUN lancia il programma dalla prima istruzione.

RUN *"U: nome file"*, R: chiude tutti i files, azzera le variabili, carica il programma richiesto e ne lancia l'esecuzione. Se non viene specificata l'opzione R i files eventualmente rimasti aperti durante l'elaborazione precedente non vengono chiusi.

SAVE *"U: nome del file"*: 'salva' sull'unità disco specificata il programma presente in memoria assegnandogli il nome indicato.

SAVE "PIPPO"	salva il programma sul disco A assegnandogli il nome PIPPO
SAVE "B: PIPPO"	salva il programma sul disco B assegnandogli il nome PIPPO.

SAVE *"U: nome file", P*: salva il programma in modo 'protetto'. Un programma salvato in questo modo non può essere listato.

SAVE *"U: nome file", A*: salva il programma in forma non compattata (ASCII), permettendo così la sua manipolazione mediante le istruzioni MERGE.

SYSTEM: rilascia il BASIC.

TRON: permette di visualizzare, in fase di esecuzione il numero corrispondente alle istruzioni realmente eseguite (TRACE).

TROFF: annulla l'effetto di TRON.

VARIABILI

Nomi

I nomi delle variabili possono essere formati sia da lettere alfabetiche che da cifre. Il primo carattere però deve essere necessariamente una lettera.

Nelle vecchie versioni del Basic, l'interprete prendeva in considerazione solo i primi due caratteri del nome e pertanto due variabili aventi i primi due caratteri del nome uguali venivano considerate dal sistema come una sola variabile. Nella versione Microsoft 5. invece si possono usare ben 40 caratteri significativi.

Sul TRS80 vi è una limitazione: i nomi delle variabili non possono contenere le parole chiave del linguaggio Basic. Così non viene accettato il nome CIFRA in quanto contiene la parola chiave IF.

Tipi

Le variabili vengono classificate in quattro tipi: *intero, reale in semplice precisione, reale in doppia precisione, alfanumerico* (stringa di caratteri).

L'appartenenza ad un certo tipo viene caratterizzata da un simbolo identificatore che viene posto alla fine del nome della variabile.
I caratteri identificatori sono rispettivamente: % ! # $

Es: X% = 5 (X% è considerato di tipo intero)

Se per una variabile non viene dichiarato il tipo, la variabile stessa viene considerata reale in semplice precisione.

Tipo	Carattere identificatore	Esempio
Intero (numeri interi compresi tra −32768 e +32767)	%	X%=123:GIORNO%=365
Reale in semplice precisione (6 cifre significative)	!	SOMMA=1234.56
Reale in doppia precisione (16 cifre significative)	#	SOMMA#=123456789.123
Alfanumerico (stringhe di caratteri di lunghezza variabile con al più 255 caratteri)	$	NOME$="ANTONIO"

Variabili aventi lo stesso nome ma appartenenti a tipi diversi vengono considerate come distinte (per esempio A% e A$).

I comandi RUN e CLEAR inizializzano le variabili assegnando il valore zero a quelle numeriche e la caratteristica di stringa vuota a quelle alfanumeriche.

Lo spazio occupato in memoria ed il tempo necessario per i calcoli aumentano con l'aumentare della precisione richiesta.

Costanti ottali ed esadecimali

Le costanti ottali sono caratterizzate dall'avere il loro valore preceduto dal carattere & o dalla coppia di caratteri &O

Esempio: PRINT &O377 ——→ 255

Le costanti esadecimali invece sono caratterizzate dalla coppia di caratteri &H posta davanti al valore.

Esempio: PRINT &HFF ——→ 255

Con le costanti ottali ed esadecimali si possono rappresentare i valori interi compresi tra − 32768 e + 32767. I numeri negativi (caratterizzati dall'avere il primo bit settato) sono espressi nella loro forma complementare.

Ottale	Decimale	Esadecimale
&O1ØØØØØ	−32768	&H8ØØØ
&O1ØØØØ1	−32767	&H8ØØ1
&O017777	−1	&HFFFF
&OØØØØØØ	Ø	&HØØØØ
&OØØØØØ1	+1	&HØØØ1
&OØ77777	+32767	&H7FFF

Le costanti ottali/esadecimali possono ad esempio essere utilizzate nelle istruzioni DATA:

```
10 DATA &H3FD1
 :
 :
50 READ X: PRINT X ————→ 16337
```

DEFINT — DEFSNG — DEFDBL — DEFSTR
(Definizione implicita di tipo)

DEF*tipo lettere:* permette di definire IMPLICITAMENTE il tipo per tutte le variabili il cui nome inizia con una delle lettere specificate, senza ricorrere all'uso dei caratteri convenzionali (% ! $)

Si può comunque ancora dichiarare per alcune variabili, il cui tipo era già stato definito implicitamente, un nuovo tipo in modo esplicito, cioè mediante l'uso dei caratteri convenzionali.
In questo caso la dichiarazione esplicita è prevalente.

DEFINT J:	tutte le variabili aventi il nome che inizia con la lettera J, il cui tipo non venga dichiarato esplicitamente, vengono considerate intere.
DEFSTR A—C:	tutte le variabili il cui nome inizia per A, B, C vengono considerate di tipo alfanumerico.
DEFINT I—N,R—T:	specifica due domini di variabili intere.

L'interprete Basic ammette la ridefinizione di tipo anche nel corso del programma mentre ciò non è ammesso utilizzando il Compilatore Basic.

Conversione di tipo

La memorizzazione di un dato in una variabile viene fatta adattando il dato al tipo di variabile che lo deve contenere.

```
10      A%=123.45
20      PRINT A%

RUN

        123
```

A#=1.001# ——→ A#=1.001
A#=1.001D ——→ A#=1.001
A#=1.001 ——→ A#=1.00100046730042 Attenzione!!!
1.1/2# ——→ 0.550000011920929
1.1#/2# ——→ 0.55

```
10      INPUT"NUMERO ";X#:PRINT X#

RUN

        NUMERO 1.001
        1.001
```

Quando si deve eseguire un'operazione aritmetica di addizione, sottrazione o moltiplicazione con operandi di tipo diverso, tutti questi vengono prima convertiti nel tipo dell'operando con precisione maggiore, e solo in seguito viene eseguita l'operazione il cui risultato verrà fornito con la medesima precisione.

Pertanto il risultato di un'operazione tra due operandi di cui uno in semplice e l'altro in doppia precisione verrà fornito in doppia precisione.

Nel caso di una divisione tra due operandi di tipo intero il risultato può essere fornito in semplice precisione. In tutti gli altri casi rimane invece valido, anche per la divisione, il criterio esposto per le altre operazioni.

```
10      A=2:B=3:R=A/B
20      A=2:B#=3:R#=A/B#
30      PRINT R,R#

RUN

        .6666667    .6666666666666667
```

Anche prima di effettuare un'operazione di confronto tra operandi di tipo diverso, il sistema li riporta tutti ad un unico tipo secondo le regole esposte in precedenza.

Effetti della conversione di tipo per un dato:

La conversione di un valore reale in un intero comporta la soppressione della sua parte decimale; la conversione da doppia a semplice precisione comporta un arrotondamento del valore con la perdita delle cifre meno significative.

Attenzione: la conversione in doppia precisione di un dato intero o in semplice precisione non comporta automaticamente il suo completamento a destra con gli zeri.

```
10      A!=1.00001
20      B#=A!
30      PRINT A!,B#

RUN

        1.00001       1.000009894371033
```

Per evitare questo inconveniente si può ricorrere alla funzione STR$ come illustrato nell'esempio.

```
10      A!=1.00001
20      B#=VAL(STR$(A!))
30      PRINT A!,B#

RUN

        1.00001     1.00001
```

ESPRESSIONI ED OPERATORI

Un'espressione è costituita o da una costante (sia essa numerica che alfanumerica) o da una singola variabile o da un'insieme di variabili e costanti legate tra loro da operatori.

Esempio: (X*2) + 4/6

Gli operatori che legano tra loro le variabili e le costanti vengono suddivisi in tre categorie:

1/ Aritmetici
2/ Relazionali
3/ Logici

Operatori aritmetici

^	Elevamento a potenza	
* /	Moltiplicazione	Divisione
+ —	Addizione	Sottrazione
MOD\	Modulo	Divisione intera

Il calcolo del valore di un'espressione aritmetica viene effettuato tenendo conto del grado di priorità degli operatori in essa contenuti.

Ad esempio 5+10/5 è uguale a 7

Per modificare l'ordine in cui devono essere eseguite le varie operazioni si utilizzano le parentesi. In tal caso le sottoespressioni racchiuse tra parentesi vengono calcolate prima.

Ad esempio (5+10)/5 è uguale a 3

X MOD Y: L'operatore MOD fornisce il resto della divisione tra due numeri interi

PRINT 100 MOD 3 ——————→ 1

X\Y L'operatore \ fornisce la parte intera della divisione tra X e Y

PRINT 100\3 ——————→ 33

X deve essere intero

Operatori relazionali

Gli operatori di tipo relazionale vengono utilizzati per confrontare tra loro due variabili o costanti. Il risultato di un'operazione di relazione è zero quando la condizione espressa dall'operatore non è verificata e −1 quando invece tale relazione è verificata.

Gli operatori di tipo relazionale trovano il loro uso più comune nelle istruzioni del tipo IF...THEN...ELSE...

```
10 IF X=0 THEN PRINT "VALORE NULLO" ELSE PRINT X
```

Gli operatori relazionali accettati dal Basic sono:

=	uguale
<>	diverso
<	minore
>	maggiore
<=	minore o uguale
>=	maggiore o uguale

Operatori logici

Vengono utilizzati per esprimere delle relazioni multiple nelle istruzioni di tipo IF...THEN...ELSE...

Il risultato di un'operazione logica può essere FALSO (0) o VERO(−1)

```
10 IF A < 0 AND B < 0 THEN PRINT  "A e B SONO NEGATIVI"
```

L'ordine di priorità con cui vengono eseguite le operazioni logiche è il seguente:

NOT AND OR XOR IMP EQV

Per comodità nella tabella seguente viene indicata con 0 la condizione FALSO e con 1 la condizione VERO. In realtà il risultato di un'operazione logica viene testato tramite un confronto con lo zero:

FALSO=0 e VERO # 0.

R1 R2

NOT (negazione)		NOT R1
1	—	0
0	—	1

AND		R1 AND R2	IMP (implicazione)		R1 IMP R2
1	1	1	1	1	1
1	0	0	1	0	0
0	1	0	0	1	1
0	0	0	0	0	1

OR		R1 OR R2	EQV (equivalenza)		R1 EQV R2
1	1	1	1	1	1
1	0	1	1	0	0
0	1	1	0	1	0
0	0	0	0	0	1

XOR(ORescIusivo)		R1 XOR R2
1	1	0
1	0	1
0	1	1
0	0	0

Nel calcolo di un'espressione contenente operatori dei tre tipi (aritmetico, relazionale, logico) l'ordine di priorità è dato dalla seguente tabella:

1/ parentesi
2/ operatori aritmetici
3/ operatori relazionali
4/ operatori logici

Esempio:

MAXNUM= – ((A>B)*A+(A<=B)*B)

dà come risultato il valore massimo tra A e B

Operazioni booleane

Le operazioni booleane sui singoli bit e la loro manipolazione all'interno di una 'parola' vengono realizzate mediante gli operatori logici AND, OR, NOT,...

Questi ultimi operano su operandi formati da gruppi di 16 bit, il cui contenuto è rappresentato tramite dei numeri interi compresi tra +32767 e –32768 di cui tali 'parole' sono la forma binaria. L'operazione logica viene poi effettuata BIT a BIT.

Esempio:

15	——→	0000000000001111		
4	——→	0000000000000100		
15 AND 4	——→	0000000000000100	——→	4

Esempio:

4	——→	0000000000000100		
2	——→	0000000000000010		
4 OR 2	——→	0000000000000110	——→	6

Esempio:

–1	——→	1111111111111111		
8	——→	0000000000001000		
–1 AND 8	——→	0000000000001000	——→	8

INPUT DA TASTIERA: INPUT-LINE INPUT-INPUT $ (1)

INPUT "messaggio", var1, var2, ...

Questa istruzione permette durante l'esecuzione di un programma di trasmettere uno o più dati battendo i relativi valori sulla tastiera del terminale.

Nell'istruzione INPUT i nomi delle variabili a cui devono essere assegnati i valori trasmessi sono separati da virgole.

Dopo la visualizzazione del 'messaggio', l'operatore digita sulla tastiera i valori da assegnare alle variabili specificate nell'istruzione di INPUT, separandoli con le virgole. I valori numerici possono essere separati tra loro anche mediante degli spazi bianchi.

Esempio:

10 INPUT "COGNOME, NOME, TELEFONO"; COGNOME$, NOME$, TELEFONO#

Dopo la visualizzazione del messaggio: cognome, nome, telefono l'operatore batte sulla tastiera:

ROSSI, MARIO, 422012

e completa l'operazione battendo il tasto RETURN. A questo punto le variabili COGNOME$, NOME$ e TELEFONO # assumono rispettivamente i valori ROSSI, MARIO, 422012.

Se in fase di INPUT l'operatore vuole assegnare alle variabili il valore nullo può battere semplicemente il tasto di RETURN (battendo solo il tasto di RETURN sul TRS80, invece, le variabili mantengono i valori che possedevano prima dell'operazione di INPUT).

Qualora, in seguito ad una richiesta di input, l'operatore non inserisce tutti i valori previsti, il sistema risponde con un doppio punto interrogativo per indicare all'operatore che deve inserire ancora dei dati.

Se in fase di input viene trasmesso un valore di tipo diverso da quello previsto (ad esempio una stringa di caratteri al posto di un valore numerico) il sistema lo rifiuta e trasmette il messaggio ??REDO (incomincia da capo).

Qualora si voglia trasmettere una stringa di caratteri contenente delle virgole, occorre racchiuderla tra virgolette.

```
10      INPUT "VIA: ";VIA$
20      PRINT VIA$

RUN

        VIA: "VIA GARIBALDI,2"
        VIA GARIBALDI,2
```

LINE INPUT "messaggio"; S$

In questo caso i caratteri di separazione (virgole, spazi bianchi, ecc.) vengono considerati come caratteri appartenenti alla stringa trasmessa.

La fine della stringa può essere comunicata solo con un RETURN (ricordiamo comunque che il numero di caratteri trasmesso non può superare 255).

INPUT$ (1): (Microsoft 5.)

Trasmette di volta in volta i caratteri battuti da tastiera senza aspettare la fine della stringa (come avviene invece per l'istruzione INPUT) e ciò permette all'operatore di controllare meglio le operazioni di input in quanto ogni carattere accettato viene di volta in volta visualizzato.

Questo tipo di operazione risulta tuttavia notevolmente lento e va usato perciò solo in casi particolari.

```
100      Y$=""                          'INIZIALIZZAZIONE DI Y$'
110      X$=INPUT$(1)                   'LETTURA DI UN CARATTERE'
120      IF ASC(X$)=13 THEN 150         'RETURN'
125      IF ASC(X$)=8 THEN Y$=LEFT$(Y$,LEN(Y$)-1):
                 PRINT X$;:GOTO 110 'RICHIESTA ARRETRAMENTO CURSORE
130      PRINT X$;                      'STAMPA CARATTERE'
140      Y$=Y$+X$: GOTO 110             'AGGIORNAMENTO STRINGA'
150 'PROSECUZIONE PROGRAMMA'
```

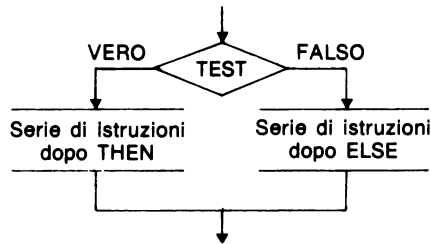
Sul TRS80 l'istruzione INKEY$ (equivalente all'INPUT$(1)) attua un esame sulla tastiera e restituisce il codice del tasto battuto, nel caso in cui un tasto sia stato effettivamente battuto, oppure una stringa vuota. Utilizzando questa funzione si può far eseguire al calcolatore un ciclo di istruzioni finchè non viene premuto un particolare tasto.

```
100      X$=INKEY$                     'ESAME TASTIERA'
110      IF X$="" THEN  GOSUB ...:GOTO 100 'SOTTOPROGRAMMA DI ATTESA'
116'
120 'ELABORAZIONE RELATIVA AL CARATTERE ACQUISITO'
125'
130       GOTO 100
```

IF...THEN...ELSE... (se...allora...altrimenti...)
Selezione e cicli

IF *espressione logica* THEN *serie di istruzioni* ELSE *serie di istruzioni*

vero o falso — se espr. è vera — se espr. è falsa

Mediante questa istruzione viene eseguito un test sul valore assunto da 'espressione logica'. Se tale valore è VERO viene eseguita la serie di istruzioni compresa tra THEN e ELSE, se è FALSO viene eseguita la serie di istruzioni che segue ELSE.

Esempio:

```
10        INPUT "NUMERO A, NUMERO B ";A,B
20        IF A>B THEN PRINT "A>B" ELSE PRINT "A<=B"
30        GOTO 10
```

In realtà il test non è limitato alle sole espressioni logiche ma può essere effettuato anche su quelle aritmetiche. In quest'ultimo caso il risultato del test sarà FALSO se l'espressione vale 0 e VERO in tutti gli altri casi.

Questa particolare opzione del Basic non verrà comunque utilizzata in quest'opera.

Esempio:

```
10        IF I THEN PRINT "I NON NULLO"
20        IF((I>10)*(I<20) THEN PRINT "I COMPRESO TRA 10 E 20"
```

Più IF possono essere concatenati tra loro. In questo caso è necessario però che ad ogni IF...THEN corrisponda un ELSE.

```
10 INPUT "A,B?";A,B
20 IF A< =B THEN IF A<B THEN PRINT "A<B" ELSE PRINT "A = B" ELSE "PRINT" A>B
```

FOR...NEXT...STEP... (ciclo)

FOR *var. contatore* = *V. iniziale TO v. finale* STEP *incremento*

:

:

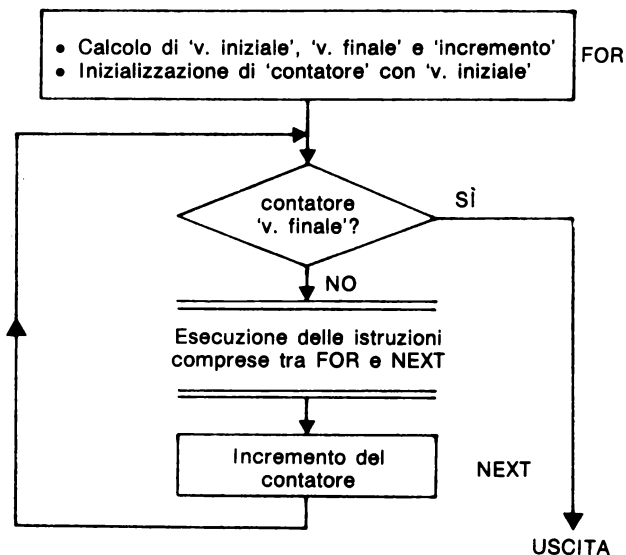
NEXT *var. contatore.*

Il 'valore iniziale', il 'valore finale' e l''incremento' possono essere assegnati mediante delle espressioni.

Quando viene incontrata un'istruzione FOR, la variabile contatore viene inizializzata con il 'valore iniziale', dopo di che vengono eseguite le istruzioni comprese tra il FOR ed il NEXT.

Allorchè viene incontrata l'istruzione NEXT, il valore della variabile contatore viene incrementato di una quantità pari a quella specificata con l'opzione STEP (se questa specifica non compare l'incremento viene assunto uguale ad 1), poi viene effettuato un confronto tra il nuovo valore assunto dalla variabile contatore e quello specificato con 'valore finale':

- se il valore del contatore risulta ancora minore o uguale (caso di un incremento positivo) al 'valore finale' vengono rieseguite tutte le istruzioni comprese tra il FOR ed il NEXT;
- se il valore del contatore risulta maggiore (caso di un incremento positivo) del 'valore limite' il programma prosegue con l'esecuzione dell'istruzione successiva al NEXT.

Nel caso in cui invece l'incremento sia negativo il ciclo ha termine quando il contatore assume un valore inferiore a quello specificato come 'valore finale'.

Esempi:

```
10      FOR I=10 TO 1 STEP -1
20          PRINT I;
30      NEXT I
40      PRINT "FINE"

RUN

        10 9 8 7 6 5 4 3 2 1 FINE
```

```
10      FOR I=0 TO 1 STEP .3
20          PRINT I;
30      NEXT I

RUN

        0 .3  .6  .9
```

I valori da assegnare ai parametri: *'valore iniziale'*, *'valore finale'* e *'incremento'* vengono calcolati una sola volta all'inizio del ciclo. Pertanto essi non cambiano anche se all'interno del ciclo vi sono delle istruzioni che modificano i valori di alcune variabili che sono state utilizzate per la loro definizione. È invece possibile modificare direttamente, tramite delle istruzioni interne al ciclo, il valore della variabile contatore.

Nella versione 5. il test sul valore assunto dal contatore viene effettuato 'in testa' al ciclo anzicchè 'in coda'. In questo caso se (per un incremento positivo) 'valore iniziale' è maggiore di 'valore finale' il ciclo non viene eseguito neppure una volta.

La 'variabile contatore' può essere sia di tipo intero che in semplice precisione. Ovviamente usando un contatore di tipo intero si rende più rapida l'esecuzione del ciclo.

Cicli annidati

Più cicli FOR...NEXT possono essere 'inscatolati' tra loro; è cioè possibile prevedere un ciclo all'interno di un altro. Non è possibile invece costruire dei cicli accavallati tra loro.

```
10      FOR I=1 TO 5                        'CICLO ESTERNO'
15          PRINT "I= ";I;" J= ";
20          FOR J=1 TO 10                   'CICLO INTERNO'
30              PRINT J;
40          NEXT J
50          PRINT
60      NEXT I
```

```
RUN

          I= 1   J= 1  2  3  4  5  6  7  8  9  10

          I= 2   J= 1  2  3  4  5  6  7  8  9  10

          :
          :

          I= 5   J= 1  2  3  4  5  6  7  8  9  10
```

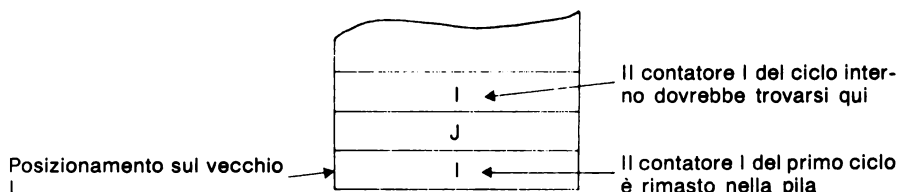
Se tra NEXT J e NEXT I non vi sono altre istruzioni si può utilizzare la forma compattata NEXT J, I.

Al posto della forma NEXT J si potrebbe utilizzare anche il solo NEXT, in quanto un'istruzione di questo tipo provoca comunque un incremento del contatore relativo al ciclo più interno. Tale contatore viene poi eliminato automaticamente al termine del ciclo stesso. Però, anche se questa forma è ammessa dal Basic da noi utilizzato, indicheremo sempre, nelle istruzioni NEXT la variabile contatore a cui si fa riferimento in quanto ciò renderà i programmi più facili da interpretare.

Nota relativa all'uso dei cicli FOR...NEXT con l'interprete Basic
(nelle versioni precedenti la 5.)

Il programma presentato nell'esempio seguente sembra a prima vista ben scritto ma posto in esecuzione non fornisce i risultati previsti.

Esempio:

```
10        FOR I=1 TO 5
20            IF I=3 THEN GOTO 40           'USCITA TRAMITE SALTO'
30        NEXT I
35'
40        FOR J=1 TO 10                  'CICLO ESTERNO'
45'
50            FOR I=1 TO 6'              'CICLO INTERNO'
60                PRINT "I= ";I
70            NEXT I
74'
75'   A QUESTO PUNTO DOVREBBE COMPARIRE IL MESSAGGIO:
76'      NEXT WITHOUT FOR
80        NEXT J
```

Il fatto si spiega nel modo seguente:

– Ogni volta che viene incontrato un FOR, la variabile contatore viene immagazzinata in una pila. Quando tale variabile raggiunge il valore limite (tramite gli incrementi relativi al NEXT), viene tolta dalla pila (viene aggiornata la posizione del puntatore alla testa della pila).

– Se però si esce da un ciclo prima che il contatore abbia raggiunto il valore limite (ad esempio mediante un GOTO), il contatore stesso viene mantenuto nella pila e, se durante l'esecuzione del programma si incontra un altro FOR che utilizza la stessa variabile come contatore, il puntatore della pila si riposiziona sul vecchio indice, che viene considerato come contatore del nuovo ciclo. In seguito a ciò, se il nuovo ciclo è un CICLO INTERNO, con il riposizionamento del puntatore alla testa della pila vengono eliminati dalla pila stessa tutti gli indici relativi ai CICLI ESTERNI a quello in esame, in quanto posizionati nella pila al di sopra del vecchio indice 1.

– Se però il nuovo ciclo non è 'annidato' con altri l'esecuzione del programma prosegue regolarmente.

Soluzione: per evitare inconvenienti di questo genere si può ricorrere al seguente artificio: utilizzare, per uscire dal ciclo prima del previsto un'istruzione del tipo:

IF...THEN I = *valore limite*: NEXT I: GOTO XX

Un'altra soluzione consiste nell'utilizzare delle variabili particolari per i cicli da cui si prevede di uscire in modo anomalo.

L'esempio seguente mostra appunto come deve essere usato questo accorgimento. Nell'esempio stesso infatti alla fine del ciclo FOR I viene eliminato dalla pila anche il contatore relativo al ciclo più interno.

```
10      FOR I=1 TO 5
20          FOR J=1 TO 4
30                 IF J=2 THEN 45
40          NEXT J
45          PRINT "J= ";J
50      NEXT I
```

VETTORI E MATRICI: DIM – ERASE

Un vettore è un insieme ORDINATO di elementi. Qualora l'ordinamento dipenda da più parametri, l'insieme viene detto matrice. La dimensione di un vettore (cioè il numero di elementi che lo compongono) viene definita tramite l'istruzione:

	VETTORE MESE		VETTORE GIORNO $
MESE (1)	31	GIORNO $ (1)	LUNEDÌ
MESE (2)	28	GIORNO $ (2)	MARTEDÌ
	• • • •		• • • •
MESE (12)	31	GIORNO $ (7)	DOMENICA

DIM Nome vet (n. elementi)

Così ad esempio il vettore MESE, che contiene il numero dei giorni relativo ai vari mesi dell'anno, viene dimensionato nel modo seguente:

```
DIM MESE (12)
```

Poichè anche i vettori possono contenere dati di diversi tipi, il tipo dei dati viene dichiarato utilizzando le stesse regole definite per le variabili di tipo scalare.

Ad esempio tramite le due istruzioni:

```
X = 7
DIM GIORNO$(X)
```

si dimensiona il vettore GIORNO di tipo alfanumerico composto da 7 elementi. (Osserviamo a questo proposito che il numero degli elementi in un'istruzione di dimensionamento può essere espresso anche tramite delle variabili. Questa forma è però ammessa solo nella versione INTERPRETE del Basic; utilizzando invece la versione COMPILATORE si possono utilizzare per il dimensionamento solo delle costanti).

Se il numero di elementi di un vettore è inferiore a 10 non è necessario dichiararne esplicitamente la dimensione. Ciononostante si consiglia di utilizzare anche in questi casi la dichiarazione esplicita DIM, sia per rendere il programma più comprensibile sia per risparmiare spazio in memoria.

Osserviamo infine che con il Basic si può utilizzare anche il valore zero per l'indice di posizionamento, anche se quest'opzione viene raramente utilizzata.

Per una matrice pluridimensionale la dichiarazione di dimensionamento viene fatta in modo simile a quello utilizzato per i vettori unidimensionali.

DIM nome matrice (dim1, dim2,...)

Consideriamo ora come esempio una matrice bidimensionale i cui elementi rappresentano il numero di vetture di un certo tipo presenti nelle filiali di una fabbrica di automobili. In particolare si sono considerati 4 modelli di vetture distribuite in 5 filiali.

MATRICE MAG

FILIALI

F. 1	F. 2	F. 3	F. 4	F. 5	
10 MAG (1,1)	5 MAG (1,2)	15 MAG (1,3)	MAG (1,4)	MAG (1,5)	M. 1
					M. 2
					M. 3
12 MAG (4,1)	10 MAG (4,2)	MAG (4,3)	MAG (4,4)	MAG (4,5)	M. 4

DIM MAG (4,5) predispone in memoria lo spazio per contenere una matrice di 4 righe e 5 colonne (20 elementi). L'inizializzazione della matrice con l'inserimento dei dati iniziali può essere realizzato nel modo seguente:

```
10        DIM MAG(4,5)
20        FOR MOD=1 TO 4
30            FOR FILIALE=1 TO 5
40                  PRINT"MODELLO: ";MOD;"   FILIALE: ";FILIALE
50                  INPUT"QUANTITA: ";MAG(MOD,FILIALE)
60            NEXT FILIALE
70        NEXT MOD


RUN


MODELLO: 1     FILIALE: 1
QUANTITA:        L'OPERATORE INSERISCE UN VALORE
MODELLO: 1     FILIALE: 2
QUANTITA:        L'OPERATORE INSERISCE UN VALORE
```

Se si vuole conoscere il numero di vetture di un certo modello presenti in tutte le filiali si può utilizzare la seguente routine:

```
100       INPUT"MODELLO: ";MOD
110       TOTALE=0
120       FOR FILIALE=1 TO 5
130           TOTALE=TOTALE+MAG(MOD,FILIALE)
140       NEXT FILIALE
150       PRINT"TOTALE VETTURE MODELLO ";MOD;" = ";TOTALE
160       GOTO 100
```

Nello stesso modo si può costruire una routine che permetta di conoscere il numero totale di vetture presenti in ogni filiale, come illustrato nell'esempio seguente:

```
200      INPUT"FILIALE: ";FILIALE
210      TOTALE=0
220      FOR MOD=1 TO 4
230          TOTALE=TOTALE+MAG(MOD,FILIALE)
240      NEXT MOD
250      PRINT"TOTALE VETTURE FILIALE ";FILIALE;" : ";TOTALE
260      GOTO 200
```

Anche per le matrici si può tralasciare la dichiarazione di dimensionamento qualora il valore massimo per ogni indice sia inferiore a 10.

È possibile utilizzare lo stesso nome per una variabile scalare ed un vettore mentre non è possibile utilizzare lo stesso nome per due vettori diversi, anche se dimensionati in modo differente.

Esempio:

Non è ammessa la forma
DIM A (7): DIM A(10,12)
per dimensionare un vettore ed una matrice distinti

ERASE nome vettore,...:	*elimina dalla memoria centrale uno o più vettori*
100 ERASE A, NOME$	elimina dalla memoria i vettori A e NOME$
110 DIM A (200)	definisce un vettore di 200 elementi

Lo spazio in memoria viene recuperato in modo dinamico. Se si richiede di cancellare un vettore non esistente viene segnalato errore.
(L'istruzione ERASE non è accettata dal compilatore Basic MBASIC 5.)

```
10'      AGGIORNAMENTO DI UNA MATRICE
20'      ============================
30'
40'      R:       RITORNO AL CAMPO PRECEDENTE
50'      RM:      RITORNO AL MODELLO PRECEDENTE
60'      RETURN:  NON VIENE ASSEGNATO ALCUN VALORE
70'
80       DATA "MILANO","TORINO","GENOVA","ROMA","NAPOLI","PALERMO"
90      FOR I=1 TO 6:READ FILIALE$(I):NEXT I
100'
110      DATA "R5","R12","R18","R20","R30","R40"
120      FOR I=1 TO 7:READ MODELLO$(I):NEXT I
125'
130      DIM MAG(7,6)
```

```
135' ------------------------------------------------------------------------
140      FOR MODELLO=1 TO 7
150          FOR FILIALE=1 TO 6
160                PRINT"MAGAZZINO: ";MODELLO$(MODELLO);"  -  ";
                        FILIALE$(FILIALE);
170                PRINT TAB(35) MAG(MODELLO,FILIALE);TAB(35);
                                              'VECCHIO VALORE'
180                INPUT X$                   'NUOVO VALORE'
190                IF X$="R" AND FILIALE>1 THEN
                      FILIALE=FILIALE-1:PRINT:GOTO 160
200                IF X$="RM" AND MODELLO>1 THEN
                      MODELLO=MODELLO-2:PRINT:GOTO 250
210                IF X$="" THEN GOTO 230
220                X=VAL(X$):IF X<0 OR X>100 THEN GOTO 160
                               ELSE MAG(MODELLO,FILIALE)=X
230          NEXT FILIALE
240          PRINT
250      NEXT MODELLO
260'
270'-----------------------------------------------------------------
280'
290'  RUN
300'
310'MAGAZZINO: R5 - MILANO          0     5
320'MAGAZZINO: R5 - TORINO          0     7
330'   :     :     :     :          :     :
```

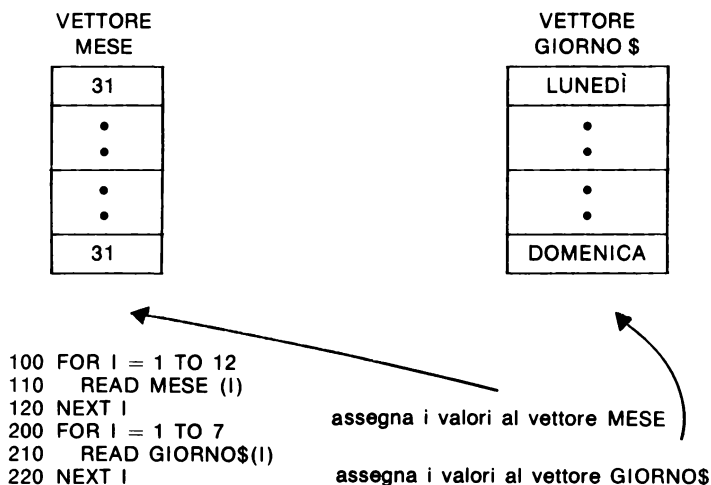
DATA — READ — RESTORE

Data valore 1, valore 2,...

L'istruzione DATA permette di definire un insieme di dati (costanti) che verranno poi utilizzati nel corso del programma. Il file definito mediante questa istruzione può essere solo letto.

```
10 DATA 31,28,30,...,31
20 DATA LUNEDÌ,...,DOMENICA
```

READ variabile

Assegna alla variabile specificata i valori contenuti nel file DATA in modo sequenziale.

Quando viene incontrata la prima istruzione READ viene assegnato alla variabile indicata il primo valore definito in DATA. Quando viene incontrata la seconda istruzione READ viene assegnato il secondo valore di DATA, e così via.

In questo modo vengono assegnati tramite le istruzioni READ i valori definiti in DATA in modo sequenziale. Qualora in un programma compaiano più istruzioni DATA, i valori in esse contenuti vengono considerati come facenti parte di un unico insieme, ordinato rispettando l'ordine con cui le istruzioni DATA sono inserite nel programma.

Ovviamente per ogni assegnazione il tipo del valore letto dal file DATA deve essere in accordo con quello dichiarato per la variabile a cui deve venire assegnato.

Bisogna inoltre fare molta attenzione durante la scrittura delle istruzioni DATA in

quanto un errore in fase di battitura, per esempio l'omissione di un dato, falserebbe poi tutta l'esecuzione del programma.

Per questi motivi in genere i valori contenuti nei DATA vengono letti nella fase iniziale del programma e trasferiti in vettori o matrici a cui si possa poi in seguito accedere in modo diretto.

Si può anche, in fase di messa a punto del programma, controllare che la lettura dei valori contenuti nei DATA venga fatta in modo corretto. Per assicurarsi di ciò è in genere sufficiente far visualizzare il valore che ogni variabile assume dopo l'assegnazione.

Come abbiamo già detto i DATA, qualunque sia la loro collocazione all'interno del programma, vengono visti, in fase di lettura, come un unico insieme ordinato di dati. L'ordine di lettura di tali dati può essere modificato solo tramite l'istruzione RESTORE.

Nei DATA i dati numerici che devono essere considerati come stringhe di caratteri devono essere racchiusi tra virgolette (").

```
DATA "3", "6", "8": FOR I= 1 TO 3: READ X$ (I): NEXT I
```

Nello stesso modo vanno racchiuse tra virgolette quelle stringhe che contengono anche caratteri di separazione

```
20 DATA "MAG:", "PIPPO, PLUTO", ...
```

RESTORE

Questa istruzione viene utilizzata quando si vuole modificare l'ordine di lettura dei vari DATA.

RESTORE *n° di riga* provoca un posizionamento del puntatore di lettura sul DATA definito nella riga specificata.

```
10      DATA 3,4,5
20      RESTORE 100          'POSIZIONAMENTO DEL PUNTATORE DI
30'                           LETTURA SULLA RIGA 100'
40'
50      READ X:READ Y:READ Z
60      PRINT X,Y,Z

100     DATA 7,8,9

RUN

        7     8     9
```

STRINGHE DI CARATTERI

. LEFT$ RIGHT$ MID$
. LEN
. STR$ VAL
. ASC CHR$
. INSTR
. STRING$ SPACE$
. OCT$ HEX$
. FRE (X$)

La parte di memoria riservata per la memorizzazione delle stringhe di caratteri è, come abbiamo già visto, o definita esplicitamente tramite il comando CLEAR o assunta automaticamente uguale a 1000 bytes o a 1/8 della capacità della memoria stessa (tra questi due valori viene utilizzato il minore).

L'assegnazione di una stringa di caratteri ad una variabile viene effettuato nel modo seguente:

nome var$ = "stringa di caratteri"

Esempio:

10 NOME$ = "ANTONIO"

Le virgolette prima e dopo ANTONIO stanno ad indicare che ANTONIO deve essere considerato come una stringa di caratteri e non come il nome di una variabile. La lunghezza di una stringa di caratteri, che non deve essere dichiarata, può variare durante l'esecuzione del programma ma non può mai superare i 255 caratteri.

Due stringhe possono essere concatenate mediante l'operatore + come illustrato nell'esempio seguente:

```
10      NOME$="PIETRO"
20      PNOME$="PAOLO"
30      NP$=NOME$+PNOME$
40      PRINT NP$

RUN

        PIETROPAOLO
```

Confronti tra stringhe

Il confronto tra due stringhe viene effettuato utilizzando gli operatori logici

= <> > < <= =>

Il confronto viene effettuato carattere per carattere, da sinistra verso destra. Quando viene incontrata una coppia di caratteri diversi verrà considerata maggiore la stringa contenente il carattere con codice ASCII maggiore.

Osserviamo che questo criterio rispetta l'ordinamento alfabetico. (GIOVANNI è maggiore di GIORGIO).

Se invece tutti i caratteri componenti le due stringhe sono rispettivamente uguali si avrà una relazione di uguaglianza.

Esempio:

```
10     INPUT"RISPONDI SI O NO";R$
20     IF R$="SI" THEN PRINT "HAI RISPOSTO SI":GOTO 60
30     IF R$="NO" THEN PRINT "HAI RISPOSTO NO":GOTO 60
40     IF R$="" THEN PRINT "PERCHE NON HAI RISPOSTO?":GOTO 10
50     PRINT "RISPOSTA NON ACCETTABILE"
60     PROSECUZIONE PROGRAMMA
```

Nota: per permettere una corretta gestione del test della riga 40, utilizzando il TRS80 occorre modificare la riga 10 nel modo seguente:

10 R$="":INPUT...

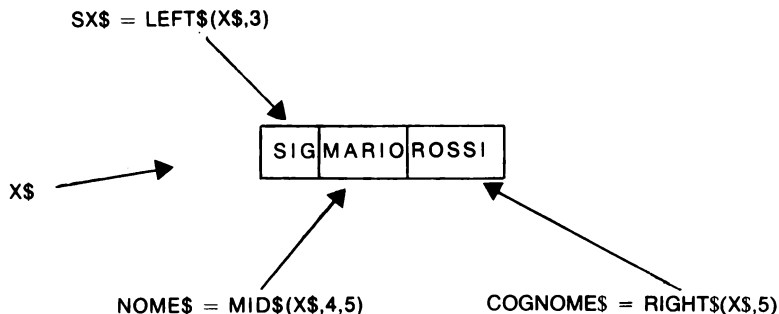
LEFT$, RIGHT$, MID$

LEFT$, RIGHT$, MID$ permettono di utilizzare solo una parte di una stringa partendo rispettivamente da sinistra, destra o da una posizione intermedia.

LEFT$(STRINGA$, *n° di caratteri a partire da sinistra)*
RIGHT$(STRINGA$, *n° di caratteri da considerare partendo da destra)*
MID$(STRINGA$, *posizione iniziale, n° caratteri)*

Tutti i parametri che rappresentano gli argomenti delle funzioni sopra descritte possono essere rappresentati anche da espressioni.

I valori di *'posizione iniziale'* e *'n° caratteri'* devono essere interi e compresi tra 0 e 255.

Esempio:

```
10      X$="SIGMARIOROSSI"
20      SX$=LEFT$(X$,3)             'TRE CARATTERI A SINISTRA'
30      NOME$=MID$(X$,4,5)          '5 CARATTERI A PARTIRE DAL QUARTO'
40      COGNOME$=RIGHT$(X$,5)       'ULTIMI 5 CARATTERI'
50      PRINT SX$,COGNOME$,NOME$

RUN

        SIG          ROSSI          MARIO
```

Esempio:

10 INPUT "RISPONDI SI O NO"; R$
20 IF LEFT$(R$, 1)= "S" THEN...
test sulla risposta dell'operatore

Esempio:

```
10      NOME$="PAOLO"
20      FOR I=1 TO 5
30         PRINT LEFT$(NOME$,I)
40      NEXT I

RUN

        P
        PA
        PAO
        PAOL
        PAOLO
```

Se il numero di caratteri specificati supera quello dei caratteri contenuti in *'stringa'*, le funzioni LEFT$, RIGHT$, MID$ restituiscono l'intera stringa.

Se nella funzione MID$ non viene considerato il parametro *'n° di caratteri'* la funzione stessa diviene equivalente alla RIGHT$.

Se *'posizione di partenza'* va oltre l'ultimo carattere della stringa, MID$ restituisce una stringa vuota.

MID$(STRINGA1$, posizione di partenza, n° caratteri da sostituire)=STRINGA 2$

Questa funzione permette di sostituire alcuni caratteri di *'STRINGA1'* con quelli contenuti in *'STRINGA2'*.

Esempio:

60 MID$("SIGMARIOROSSI", 4,5)="PAOLO" ——→ SIGPAOLOROSSI

Attenzione: questa funzione non permette nè l'inserimento nè la cancellazione di caratteri ma solo la loro sostituzione.

Se *STRINGA2* contiene più caratteri di quelli che si intendono sostituire in *STRINGA1*, la sostituzione viene effettuata utilizzando i caratteri necessari partendo dalla sinistra di *STRINGA2*. Se invece *STRINGA2* contiene meno caratteri di quelli necessari, viene effettuata solo una sostituzione parziale.

```
10 MID$("123456",3,2) = "XXX" ------→ 12XX56
20 MID$("123456",3,2) = "X"   ------→ 12X456
```

LEN (stringa)

Questa funzione calcola il numero di caratteri contenuti in una stringa.

```
10      NOME$="ANTONIO"
20      L=LEN(NOME$)
30      PRINT L

RUN

        7
```

STR$ (X)

Converte il valore numerico X nella stringa di caratteri corrispondente.

```
10      X=123
20      X$=STR$(X)
30      PRINT X$,LEN(X$)

RUN

        123       4
```

Nota: il primo carattere a sinistra della stringa generata dalla funzione STR$ è riservato al segno ed è uno spazio bianco se X è positivo e un – se X è negativo.

```
10X=-1
20 PRINT MID$(STR$(X),2,1) ------→ 1
```

VAL (stringa)

È la funzione inversa della precedente e trasforma una stringa di caratteri (cifre) in un numero.

```
10      X$="125 LIRE"
20      X=VAL(X$)
30      PRINT X

RUN

        125
```

Se il primo carattere della stringa non è nè una cifra, nè uno spazio bianco, nè un +, nè un – la funzione restituisce il valore zero.

VAL(" 123") ———→ 123
VAL(" +123") ———→ 123
VAL("ABC") ———→ 0
VAL("A+12") ———→ 0

ASC (carattere)

Restituisce il valore numerico del codice ASCII del carattere specificato.

```
10        X$="A"
20        X=ASC(X$)
30        PRINT X

RUN

          65
```

ASC (stringa)

Restituisce il codice ASCII del primo carattere della stringa.

PRINT ASC("BUONGIORNO") ———→ 66

Viene segnalato errore se l'argomento della funzione è una stringa vuota.

CHR$(X)

È la funzione inversa della precedente e restituisce il carattere avente il codice ASCII corrispondente al valore di X.
X, che può essere espresso tramite una costante o una variabile oppure una espressione, deve essere intero e compreso tra 0 e 255.

```
10        FOR I=65 TO 65+26
20            PRINT CHR$(I);
30        NEXT I
40'
50'  RUN
60'
70'       ABCDE..........Z
```

Conversione MAIUSCOLO – minuscolo
PRINT CHR$(ASC("A")+32) ———→ a (cfr. tabella dei codici ASCII)

```
10        INPUT"INSERITE IL VOSTRO NOME: ";NOME$
20        FOR I=1 TO LEN(NOME$)
30            PRINT CHR$(ASC(MID$(NOME$,I,1))+32);
40        NEXT I
```

RUN

```
INSERITE IL VOSTRO NOME: ANTONIO

antonio
```

Caratteri speciali (di controllo)

PRINT CHR$(7) ———→ attiva il segnale acustico

PRINT CHR$(13) ———→ provoca il ritorno all'inizio della riga (senza salto di riga)

PRINT CHR$(10) ———→ provoca un salto di riga

La funzione CHR$(X) è notoriamente utilizzata per inviare dei segnali di controllo alle periferiche (terminale, stampante, plotter,...). Il vantaggio di questa funzione consiste nel permettere la visualizzazione di tali caratteri di controllo nel listing del programma.

Utilizzando invece i tasti di controllo, posti direttamente sulla tastiera, per definire delle istruzioni da inserire in un programma, qualora tale procedura fosse permessa si otterrebbero questi due effetti:

– in fase di list i caratteri di controllo utilizzati verrebbero trasmessi al terminale come segnali di controllo da interpretare. Ad esempio un "Control G" (codice ASCII = 7) provocherebbe un segnale sonoro.

– i caratteri interpretati come segnali di controllo non verrebbero inseriti nel listing.

Esempi diversi

Cancellazione dell'ultimo carattere di una stringa

```
10        X$="DOPPIO"
20        X$=LEFT$(X$,LEN(X$)-1)
30        PRINT X$

RUN

          DOPPI
```

Troncamento di una stringa o sua normalizzazione su una lunghezza prefissata

```
10        X$="PIPPO"
20        Y$=RIGHT$("          "+X$,10)
30        PRINT "A";Y$, LEN(Y$)

RUN

          A     PIPPO          10
```

Inserimento di un carattere in una stringa

```
10        Z$="AAAAAAA"
20        X$="B":L=3.
30        Z$=LEFT$(Z$,L)+X$+RIGHT$(Z$,LEN(Z$)-L)
40        PRINT Z$

RUN

          AAABAAAA
```

Completamento di una stringa con una serie di zeri a sinistra

```
10        X=123
20        X$=RIGHT$(STR$(100000+X),5)
30        PRINT X$

RUN

          00123
```

INSTR (posizione di partenza, stringa, stringa cercata)

Ricerca la posizione di una determinata stringa in un'altra a partire da 'posizione di partenza'. Quest'ultimo parametro, quando non è specificato, viene assunto automaticamente uguale a 1.

- Se la stringa cercata non viene trovata, la funzione assume il valore 0.
- Se invece essa è vuota, la funzione assume il valore specificato in 'posizione di partenza'.
- Se 'posizione di partenza' oltrepassa l'ultimo carattere di 'stringa' la ricerca non viene effettuata e la funzione assume il valore zero.

Es.

```
10        X$="ROSSI.MARIO"
20        X=INSTR(X$,".")              'RICERCA DEL CARATTERE
30        Y$=LEFT$(X$,X-1)
40        PRINT Y$,X

RUN

          ROSSI          6
```

Es.: PRINT INSTR (3,"ABAABAAA", "B") ——————→ 5

Esempi d'applicazione della funzione INSTR

Conversione esadecimale - decimale

```
10        X$="C"
20        X=INSTR("0123456789ABCDEF",X$)-1
30        PRINT X

RUN

          12
```

Conversione alfabetico - numerica

La routine presentata permette di utilizzare dei codici operativi di tipo letterale anzichè numerico (l'utente li ricorda con maggior facilità).

```
10        INPUT"INSERIRE CODICE OPERATIVO: ";COD$
20        X=INSTR(" CABMPQ",COD$) 'RICERCA POSIZIONE DEL CODICE'
30        ON X GOTO 10,200,300,500,650,700,750      'SALTO ALLA ROUTINE
                                                      INTERESSATA'
40        GOTO 10
```

Nota: Se COD$ è una stringa vuota si ha X=1. Per questo motivo il primo carattere della stringa utilizzata nell'istruzione 20 è stato posto uguale ad uno spazio bianco.

Verifica su un carattere

Questa routine permette di verificare se il carattere battuto sulla tastiera appartiene ad un insieme di caratteri accettabili (nell'esempio le cifre decimali ed il punto).

```
10        X$=INPUT$(1)                  'LETTURA DI UN CARATTERE'
20        X=INSTR("0123456789.",X$      'RICERCA DEL CARATTERE'
30        IF X=0 THEN GOTO 10           'CARATTERE RIFIUTATO'
40        Y$=Y$+X$                      'ACQUISIZIONE DEL CARATTERE'
50        PRINT X$;                     'VISUALIZZAZIONE DEL CARATTERE'

100       GOTO 10
```

STRINGS$ (n. volte, stringa)

Genera una stringa di caratteri uguale a quella specificata, ripetuta n volte.

```
10        X$=STRING$(10,".")            'GENERAZIONE DI 10 PUNTI'
20        PRINT X$

RUN

          ..........
```

Il programma seguente completa delle stringhe con dei punti a destra:

```
10      NZ$(1)="COGNOME"
20      NZ$(2)="NOME"
30      NZ$(3)="DATA DI NASCITA"
40      FOR I=1 TO 3
50          PRINT NZ$(I);STRING$((20-LEN(NZ$(I))),".");
60          INPUT X$(I)
70      NEXT I

RUN

        COGNOME.............ROSSI
        NOME................MARIO
        DATA DI NASCITA.....21/7/1950
```

SPACE$ (X)

Genera una stringa formata da X spazi bianchi. X può essere anche un'espressione.

```
10      X$="MARIO"+SPACE$(10)+"ROSSI"
20      PRINT X$

RUN

        MARIO          ROSSI
```

Osservazione: la funzione SPC, che a differenza della SPACE$ non genera alcuna stringa, può essere usata solo in corrispondenza di un PRINT.

OCT$ (espressione)

Genera una stringa di caratteri che rappresenta il valore ottale dell'espressione interessata.

PRINT OCT$(24) ——————→ 30

HEX$ (espressione)

Genera una stringa di caratteri che rappresenta il valore esadecimale dell'argomento decimale.

PRINT HEX$(26) ——————→ 1A

FRE (X$)

Fornisce lo spazio ancora disponibile in memoria per memorizzare delle stringhe di caratteri (X$ è un argomento fittizio)

PRINT FRE(X$) ——————→ 1000.

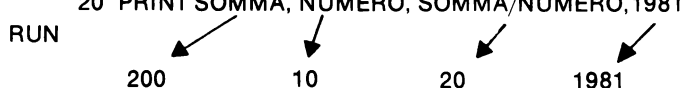
STAMPE

PRINT	POS
TAB	LPRINT
SPC	WRITE
PRINT USING	WIDTH

La stampa dei risultati non è certo la parte più nobile dell'informatica ma riveste tuttavia una certa importanza. Infatti ogni volta che si deve stampare una serie di dati è indispensabile predisporre un tabulato che renda i risultati di facile consultazione.

PRINT espressione

L'istruzione PRINT seguita da una costante, da una variabile o da un'espressione genera la visualizzazione sul terminale (sia esso video o a stampa) del valore di quella costante (o variabile o espressione) e il riposizionamento del cursore all'inizio della riga successiva.

```
10      X=123
20      PRINT X

RUN

        123
```

PRINT

La visualizzazione di più valori su una stessa riga può essere ottenuta semplicemente separando i nomi delle variabili, nell'istruzione PRINT, mediante delle virgole.

Utilizzando la virgola come carattere separatore si ottiene la visualizzazione dei dati in campi prestabiliti (con inizio ad esempio alla colonna 1, 15, 29, ...). Ciascun dato viene inserito nel campo partendo da sinistra.

PRINT

Un punto e virgola alla fine dell'istruzione PRINT inibisce il riposizionamento a capo del cursore. Inoltre, nel caso in cui l'istruzione PRINT faccia riferimento a più variabili, utilizzando come carattere separatore il punto e virgola si ottiene il seguente

effetto: le stringhe di caratteri stampate vengono concatenate, mentre i valori numerici vengono separati tra loro da uno spazio bianco.

```
10      NOME$="MARIO";COGNOME$="ROSSI"
20      PRINT NOME$;SPC(3);COGNOME$

RUN

        MARIO   ROSSI
```

PRINT "ROSSI"; " "; "MARIO" ———→ ROSSI MARIO
PRINT 123;456; −789 ———→ 123 456 −789

PRINT TAB (X)

La funzione TAB(X) (X può essere un'espressione) permette di posizionare, in fase di stampa, il cursore sulla colonna X.

```
    10   A=123:B=456
    20   PRINT A; TAB (15) B
RUN
    123          456
                 ↑
                 15 colonna
```

Attenzione: il cursore o la testina di scrittura non possono retrocedere. Pertanto la funzione TAB (X) è priva di effetto se la colonna indicata da X è precedente a quella su cui è posizionato in quel momento il cursore.

PRINT SPC(X)

La funzione SPC genera X spazi bianchi a partire dalla posizione occupata dal cursore.

```
10      NOME$="MARIO":COGNOME$="ROSSI"
20      PRINT NOME$;COGNOME$

RUN

        MARIOROSSI
```

PRINT USING # . + − ** $$ **$ _ % \\ ! &

L'istruzione PRINT USING è senza dubbio il mezzo più potente per organizzare dei tabulati in BASIC.

Variabili numeriche: Come abbiamo visto SENZA il PRINT USING I VALORI NUMERICI vengono inseriti nella parte SINISTRA del campo, mentre per la maggior parte delle applicazioni occorrerebbe un INCOLONNAMENTO A DESTRA.

PRINT USING "#####"; espressione numerica

Il campo è definito dal numero di # (ciascun # rappresenta una cifra) e l'incolonnamento viene fatto partendo da destra.

```
10        X=123:Y=1234
20        PRINT USING"#####";X
30        PRINT USING"#####";Y

RUN

            123
           1234
```

Utilizzando invece la semplice istruzione PRINT si sarebbe ottenuto:

```
123
1234
```

Nella definizione del campo la serie di # può essere rappresentata tramite una variabile alfanumerica.

```
10        X=123:Y=1234
20        FMT$="#####"
30        PRINT USING FMT$;X
40        PRINT USING FMT$;Y

RUN

            123
           1234
```

PRINT USING "###.#"; espressione numerica

Il numero di # posti dopo il quarto sta ad indicare la precisione con cui deve essere stampato il valore dell'espressione (cioè il numero di cifre dopo la virgola).

```
    10  X=123.456                    2 cifre dopo la virgola

    20  PRINT USING "####.##";X
RUN
        123.46
```

Come si vede il valore da stampare viene arrotondato automaticamente.

FORMAT MULTIPLO: PRINT USING "format1 format2"; esp1, esp2

Con questa istruzione è possibile associare ad ogni valore da stampare un format differente

```
      10  X=123.456:Y=67
      20  PRINT USING "####.##   ##.##"; X;Y
RUN
          123.46  67.00
```

SPECIFICHE: PRINT USING "spec1 format1 spec2 format2"; esp1, esp2

In fase di stampa vengono inseriti i messaggi specificati.

Se si deve usare un unico format per tutte le variabili si può usare anche la forma:

```
10      X=123:Y=456:Z=789
20      PRINT USING "####";X,Y,Z

RUN

        123 456 789
```

\+ il segno "+", se non è specificato nel format non viene visualizzato:

```
 5  PRINT USING " ####";   123 ------>   123
10  PRINT USING "+####";   123 ------>  +123
20  PRINT USING "+####";  -123 ------>  -123
```

— un segno "–" alla fine del format provoca la stampa, in caso di valore negativo, del segno "–" dopo il numero anzichè prima.

```
10  PRINT USING "####-"; -325 ------>  325-
20  PRINT USING "####-";  325 ------>  325
```

** 2 asterischi all'inizio del format, oltre ad ampliare il campo di due spazi, provocano in fase di stampa il completamento a sinistra del campo con degli asterischi.

```
10  PRINT USING "**####"; 123    ------→  ***123
20  PRINT USING "**####"; 1234   ------→  **1234
30  PRINT USING "**####"; 12345  ------→  *12345
```

$$ 2 "$" all'inizio del format provocano, in fase di stampa, l'aggiunta di un "$" alla sinistra del valore da stampare.

```
10  PRINT USING "$$####"; 23 ------→  $23
```

$ combina gli effetti di "" e "$$"

```
10  PRINT USING "**$####.##"; 2.34 ------→  *****$2.34
```

% se il campo prefissato è insufficiente per contenere il valore da stampare, il carattere % viene stampato davanti al valore stesso

```
10  PRINT USING "###"; 12345 ------→  %12345
```

_ il carattere "_" definisce come carattere di commento quello che lo segue nel format. Con questo accorgimento si possono usare in commento anche dei caratteri funzionali.

```
10  PRINT USING "_.A=####"; 123 ------→  .A= 123
```

Stringhe di caratteri

PRINT USING "\ \"; stringa

Il numero di spazi compreso tra le due "\" definisce il NUMERO DI CARATTERI di stringa −2 da stampare.

```
10        X$="RENAULT"
20        PRINT USING "\      \";X$

RUN

          RENAU
```

PRINT USING "!"; stringa

Provoca la stampa del primo carattere della stringa

PRINT USING "&"; stringa

Permette di non definire l'ampiezza del campo. Il campo viene definito automaticamente dalla lunghezza della stringa.

```
10        COGNOME$="ROSSI":NOME$="MARIO"
20        PRINT USING "&  !";COGNOME$,NOME$

RUN

          ROSSI          M
```

Stampa combinata di valori numerici e alfanumerici

Si possono definire mediante un unico PRINT USING i formats relativi a valori numerici ed alfanumerici:

Programma per il controllo dei dati

Il programma seguente permette di comunicare in fase di input, oltre al dato che il format con cui dovrà essere poi stampato.

POS (0)

Restituisce la colonna sui cui è posizionato in quel momento il cursore (l'argomento 0 non viene utilizzato).

```
100       PRINT"XXXX";:A=POS(0):PRINT
110       PRINT TAB(A) "YYYY"

RUN

          XXXX
              YYYY
```

LPOS(0)

Dà la posizione della testina di scrittura della stampante.

```
100 IF LPOS(0) > 80 THEN LPRINT
```

LPRINT

Questa istruzione genera l'output su stampante. Con essa si possono utilizzare tutte le opzioni ammesse per il PRINT.

WRITE esp1, esp2, ... (versione 5.)

Ha la stessa funzione del PRINT... ma separa i singoli valori con delle virgole e stampa le stringhe tra virgolette.

```
10        A=80:B=90:C$="ALFA"
20        WRITE A,B,C

RUN

          80,90,"ALFA"
```

WIDTH lunghezza (versione 5.)

Precisa il numero massimo di caratteri da visualizzare su una riga del video (se non si utilizza la funzione WIDTH tale numero è 72). Nel sistema Basic, una volta raggiunto il limite massimo di caratteri per una riga, si ha il riposizionamento automatico del cursore all'inizio della riga successiva.

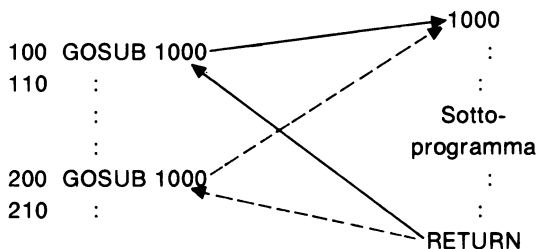
Il numero massimo di caratteri, che può essere specificato anche tramite un'espressione intera, deve essere compreso tra 15 e 255. Con il valore 255 si definiscono stringhe di lunghezza 'infinita'. Utilizzando quest'ultima opzione la funzione WIDTH permette di inviare dalla tastiera un qualunque numero di caratteri speciali di controllo senza il pericolo che il Sistema Operativo inserisca automaticamente dei RETURN (CR) in modo inopportuno. Inoltre il valore di POS(0) viene riposizionato a zero dopo la battitura di 255 caratteri.

```
100   WIDTH   120
```

WIDTH LPRINT n

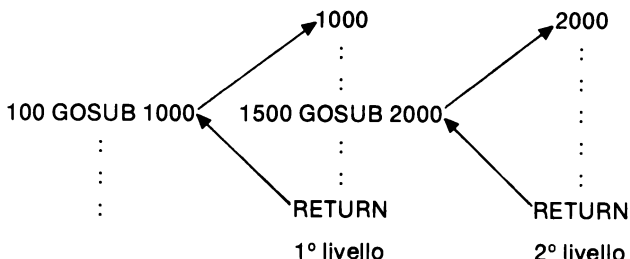
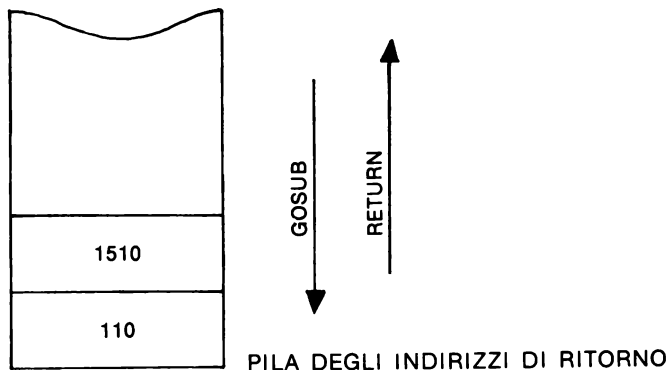
Viene utilizzato per la stampante.

SOTTOPROGRAMMI GOSUB – RETURN

Se si deve inserire più volte la stessa sequenza di istruzioni nel corso di un programma, si può ricorrere alla struttura di sottoprogramma che permette di scrivere tali istruzioni una volta sola.

Un sottoprogramma Basic è formato da un insieme di istruzioni, inserite nel programma principale, che seguono la numerazione del programma. Il sottoprogramma viene richiamato tramite un'istruzione GOSUB xx. Quest'istruzione genera nell'esecuzione del programma un salto all'istruzione xx (come se fosse un GOTO). Quando poi, proseguendo l'esecuzione si incontra un RETURN, si ha un nuovo salto all'istruzione immediatamente successiva al GOSUB di chiamata.

Un sottoprogramma può a sua volta chiamarne un altro, e così di seguito. Per assicurare il corretto ritorno nel caso di sottoprogrammi concatenati, il sistema opera nel modo seguente:

Ogni volta che viene eseguita un'istruzione di GOSUB, l'INDIRIZZO DI RITORNO (indirizzo dell'istruzione successiva al GOSUB stesso) viene immagazzinato in una pila o STACK (struttura LIFO: Last in First Out) e, ogni volta che viene eseguito un RETURN l'indirizzo di ritorno viene prelevato dalla testa della pila stessa.

Per questo motivo non conviene entrare o uscire da un sottoprogramma mediante dei salti incondizionati del tipo GOTO.

Se si entra in un sottoprogramma tramite un GOTO, il sistema segnalerà errore solo se al momento del RETURN troverà la pila degli indirizzi vuota.

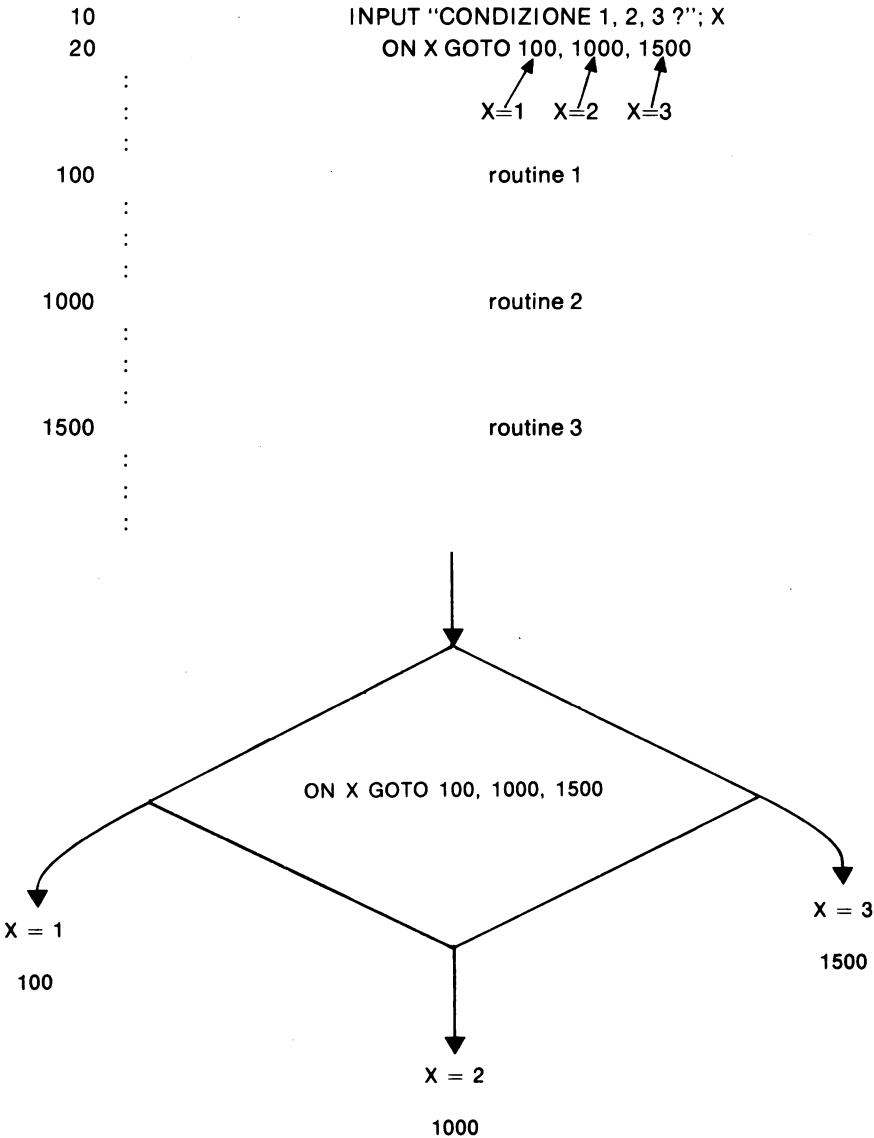
La scelta della struttura a sottoprogramma non è dettata solo dal desiderio di risparmiare spazio in memoria ma anche, e soprattutto, per le seguenti ragioni:

- utilizzando i sottoprogrammi si ottiene un programma più articolato e quindi più comprensibile;
- l'uso dei sottoprogrammi permette di ripartire il lavoro di programmazione tra più persone;
- la messa a punto può essere fatta per un sottoprogramma alla volta; in questo modo risulta notevolmente facilitato il lavoro;
- non utilizzando i sottoprogrammi, oltre a dover scrivere più volte le stesse sequenze di istruzioni, si dovrebbero riportare più volte le eventuali correzioni che si rendessero necessarie in fase di messa a punto (cosa piuttosto noiosa) e si rischierebbe quindi di rimettere in discussione l'intero lavoro;
- utilizzando i sottoprogrammi si può facilmente modificare la struttura dell'intero programma. È infatti possibile aggiungere o togliere delle routines dal programma modificando solo le istruzioni di chiamata.

I vantaggi derivanti dall'uso dei sottoprogrammi sono indubbiamente notevoli. Pertanto consigliamo di utilizzarli ogni qualvolta se ne presenti l'opportunità.

ON...GOTO... — ON...GOSUB...

ON X GOTO n1, n2,...
(test multiplo su X)

Questa funzione provoca un salto alla linea n1 o n2 o n3... rispettivamente nei casi in cui X assume il valore 1 o 2 o 3...

Se il valore di X non è intero viene arrotondato
Se X<0 o X>255 si ha segnalazione di errore
Se ad un valore di X non corrisponde alcun numero di linea o se X=0 si ha un salto alla riga successiva.

Tenendo conto di ciò assume significato il seguente programma:

```
10      ON X GOTO 100,1000,1500                  '(X=1,2,3)'
20      ON (X-5) GOTO 200,300,400                '(X=6,7,8)'
30      PRINT X;"VALORE NON ACCETTABILE"         '(ALTRI VALORI)'
40      STOP
```

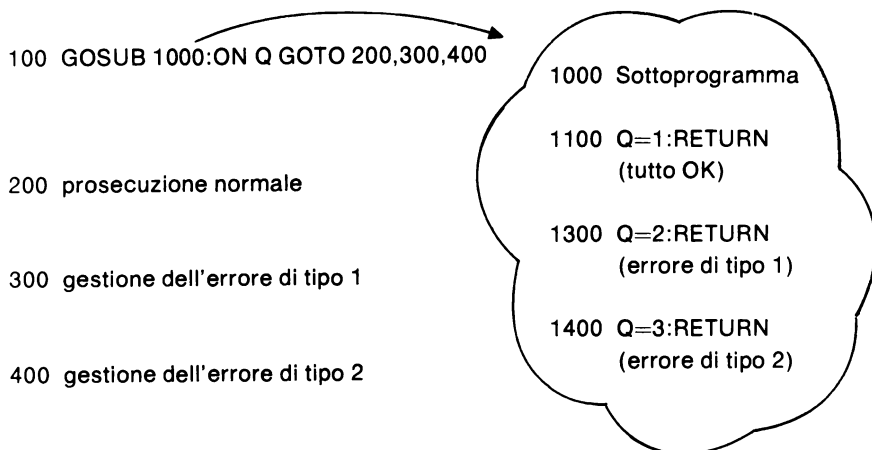
ON X GOSUB n1, n2,...

Si ha un salto al sottoprogramma avente inizio alla *riga n1 o n2...* rispettivamente nel caso in cui X assume il valore 1 o 2...

In ogni caso il RETURN provoca un ritorno all'istruzione successiva a quella di chiamata ON.

```
  10 ON X GOSUB 100, 1000, 1500
  20 .....                          istruzione a cui tornano i
   :    :                           vari sottoprogrammi
 100 ...                            sottoprogramma 1
   :    :
 190 RETURN
   :    :
1000 ...                            sottoprogramma 2
   :    :
1050 RETURN
   :    :
1500 ...                            sottoprogramma 3
   :    :
1650 RETURN
```

Test sulle condizioni all'uscita da un sottoprogramma

Nei sottoprogrammi si utilizza spesso un parametro, detto parametro "spia", che, passato al programma principale, rivela la presenza o meno di alcune condizioni particolari intervenute durante l'esecuzione del sottoprogramma. Per testare questo tipo di parametro risulta particolarmente efficace la forma del tipo ON Q GOTO... come illustrato nell'esempio.

ON ERROR GOTO.. – ERR – ERL – RESUME.. – ERROR..
(Trattamento di errori)

Utilizzando l'istruzione ON ERROR, ogni qualvolta viene riscontrato un errore in fase di esecuzione del programma si provoca un salto ad una apposita routine per il trattamento degli errori.

Tale routine analizza l'errore avvenuto testando i valori di ERR ed ERL, in cui il sistema ha posto il codice dell'errore, ed il n° di linea in cui l'errore si è prodotto.

Dopo il riconoscimento ed il trattamento dell'errore mediante l'istruzione RESUME si può far riprendere il programma da un particolare punto.

```
10      ON ERROR GOTO 100
20      INPUT "DIVISORE ";X
30      PRINT 100/X
40      GOTO 20

100     IF ERR=11 AND ERL=30 THEN PRINT"IL DIVISORE O NON AMMESSO":
            RESUME 20
110     PRINT "ERRORE NON RICONOSCIUTO":STOP
```

ON ERROR GOTO 0

Inserito nella routine per il trattamento degli errori annulla l'effetto dell'istruzione ON ERROR GOTO.. e passa il controllo al sistema.

RESUME

Scritto alla fine della routine per il trattamento degli errori indica da dove deve essere ripreso il programma principale.

RESUME:
o RESUME 0 — l'esecuzione riprende dall'istruzione che aveva generato l'errore.

RESUME NEXT: l'esecuzione riprende dall'istruzione successiva a quella che ha provocato l'errore.

RESUME n: l'esecuzione riprende dalla riga specificata.

ERROR

ERROR n: permette all'utente di definire dei propri codici di errore (compresi tra 0 e 255) e provoca un salto all'istruzione ON ERROR... come se il sistema avesse riconosciuto un errore.

10 ERROR 153: provoca un salto ad ON ERROR...

Nota: L'istruzione ON ERROR GOTO... viene generalmente posta in testa al programma in modo che venga riconosciuta dal sistema prima che sopraggiunga qualche errore.

FUNZIONI

Funzioni aritmetiche

L'argomento X di queste funzioni può essere una costante, una variabile o una espressione aritmetica.

ABS(X) Fornisce il valore assoluto di X.
100 PRINT ABS (–35) ——→ 35

ATN(X) Fornisce l'arcotangente di X, espresso in radianti, (valori compresi tra $-\pi/2$ e $\pi/2$)

CDBL(X) Converte X in doppia precisione (16 cifre significative). Ciò permette di effettuare una operazione in doppia precisione.
100 FOR I% =1 TO 10: PRINT 1/CDBL(1%): NEXT 1%

CINT(X) Converte X in un intero mediante un arrotondamento
100 PRINT CINT(1.6) ——→ 2
110 PRINT CINT(–1.2) ——→ –1
(X deve essere compreso tra –32768 e 32767)

COS(X) Fornisce il valore del coseno di X (X in radianti)

CSNG(X) Converte X in semplice precisione (6 cifre significative)
100 X=1234.56789123:A=CSNG(X):PRINT A ——→ 1234.57

EXP(X) Fornisce la funzione esponenziale di X.

FIX(X) Sopprime le cifre dopo il punto decimale.
100 PRINT FIX(2.2) ——→ 2
110 PRINT FIX(–2.6) ——→ –2

FRE(0) Restituisce il numero di bytes disponibili in memoria (l'argomento non viene utilizzato)

INT(X) Fornisce la parte intera di X.
100 PRINT INT(2.2) ——→ 2
110 PRINT INT(–2.2) ——→ –3

LOG(X) Fornisce il logaritmo naturale di X.

RANDOMIZE (non è una vera e propria funzione). Viene utilizzata per inizializzare il generatore di numeri casuali. Quando, in fase di esecuzione, viene incontrata questa funzione, viene richiesto all'utente di inserire un valore a piacere (compreso in un certo campo) e mediante questo valore viene inizializzato il generatore di numeri casuali. Senza que-

sto accorgimento la serie di numeri casuali generata risulterebbe sempre uguale.

```
10  RANDOMIZE
20  FOR I=1 TO 5: PRINT RND(1);:NEXT I
RUN
Random Number Seed (0–65529)? 3(3 è la risposta dell'operatore)
.88598 .484668 .586328 .119426 .709225
```

RND(X)

X>0 o senza X ——————→ numero casuale seguente
X=0 ——————→ stesso numero casuale
X<0 ——————→ ricomincia la sequenza

Es.: FOR I=1 TO 5: PRINT INT(RND(1)*100);:NEXT I
RUN
24 30 31 51 1

Sul TRS80 la funzione RDN(X) ha il seguente effetto:

0<=X<1 —————→ numero casuale compreso tra 0 e 1
1<X<32767—————→ numero casuale intero compreso tra 0 e X (viene considerata solo la parte intera di X).

SGN(X) Fornisce uno dei tre valori +1, –1, 0, in relazione al segno di X e rispettivamente:

+1 se X>0
0 se X=0
–1 se X<0

SIN(X) Fornisce il seno di X (X in radianti).

SQR(X) Fornisce la radice quadrata di X (X non negativo).

TAN(X) Fornisce la tangente di X (X espresso in radianti).

DEFINIZIONE DI FUNZIONI DEF FN

Oltre ad utilizzare le funzioni di sistema (SQR; SGN, INT,...) l'utente può definire direttamente delle proprie funzioni mediante l'istruzione:

DEF FNXX(X,Y,Z,...)=espressione (X,Y,Z,...)

ove XX rappresenta il nome simbolico della funzione (per comporre il nome di una funzione si utilizzano le stesse regole valide per i nomi delle variabili) e X, Y, Z,... gli argomenti della funzione stessa.

Nota: Su alcune versioni del Basic il numero di argomenti può essere al massimo 2.

La funzione verrà poi richiamata dal programma assegnando dei valori reali ai parametri.

Esempi

Funzione di arrotondamento

```
10        DEF FNAR(X)=INT(X+0.5)
20        A=123.6:B=345.2
30        PRINT FNAR(A)
40        PRINT FNAR(B)

RUN

          124
          345
```

Funzione per la ricerca del massimo tra due valori

```
10        DEF FNMX(X,Y)= -((X>Y)*X + (X<=Y)*Y)
20        A=15:B=20                |          |
30        PRINT FNMX(A,B)          |___       |___
40                                  ASSUMONO IL VALORE 0 OPPURE -1

RUN

          20
```

Nota: Ricordiamo che il risultato delle operazioni logiche X>Y ed X<=Y è 0 quando le disuguaglianze non sono verificate e –1 in caso contrario.

Funzione di arrotondamento al centesimo

```
10        DEF FNAC(X)=INT(X*100 + 0.5)/100
20        PRINT FNAR(123.456)

RUN

          123.46
```

Note:

– una funzione deve essere obbligatoriamente scritta su di un'unica riga di programma; per la sua definizione quindi non si possono usare più di 255 caratteri.

– si consiglia di definire le funzioni in testa al programma onde evitare che vengano richiamate prima della loro definizione. In tal caso infatti l'istruzione di chiamata verrebbe interpretata in tutt'altro modo.

– una stessa funzione (stesso nome) può essere ridefinita più volte all'interno del programma.

SWAP var1, var2 (versione 5.)

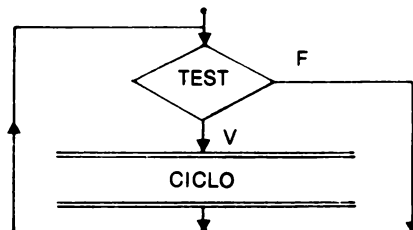
Questa funzione provoca lo scambio di valori tra le due variabili, che ovviamente devono essere dello stesso tipo.

```
100     A(1)=10:A(2)=20
110     PRINT A(1),A(2)
120     SWAP A(1),A(2)
130     PRINT A(1),A(2)

RUN

        10        20
        20        10
```

WHILE...WEND (versione 5.)

WHILE *espressione logica*
:
:
WEND

Utilizzando questa struttura tutte le istruzioni comprese tra WHILE e WEND vengono eseguite ciclicamente finchè l'espressione logica è VERA. Nel momento in cui questa condizione non è più verificata, il programma prosegue dall'istruzione successiva a WEND.

Osservazione: l'espressione che segue WHILE può essere anche aritmetica: in tal caso il ciclo viene eseguito finchè il valore dell'espressione diventa zero.

```
10 INVERSIONE=1
20 WHILE INVERSIONE ><0
30   INVERSIONE=0
40   FOR I=1 TO 9
50     IF A(I)>A(I+1) THEN SWAP A(I), A(I+1): INVERSIONE=1
60   NEXT I
70 WEND
```

Si potrebbe simulare la struttura WHILE con un ciclo FOR nel modo seguente:

```
10 FOR I=1 TO 'infinito'
20       IF esp =0 THEN GOTO USCITA
                       :
30 NEXT I              :
USCITA        .......
```

CHAIN COMMON (MICROSOFT 5.) (CHIAMATA DI PROGRAMMI)

CHAIN MERGE nome file, insert, ALL, DELETE linea1 - linea2

Chiama un programma trasmettendo le variabili del programma corrente.

```
100 CHAIN "PIPPO"
```

L'opzione 'insert' specifica la linea da cui inizierà l'esecuzione del programma chiamato.

```
100 CHAIN "PIPPO", 1000
```

L'opzione ALL specifica che tutte le variabili del programma corrente vengono passate in blocco al programma chiamato.

L'opzione MERGE viene utilizzata qualora si desideri che una parte del programma chiamato 'ricopra' alcune linee del programma corrente. Perchè ciò sia possibile è necessario che i programmi siano memorizzati in ASCII.

```
100 CHAIN MERGE "PIPPO",1000
```

Una parte del programma può anche essere soppressa mediante l'opzione DELETE.

```
100 CHAIN "PIPPO",1000,DELETE 2000-2500
```

(il compilatore BASIC non accetta le opzioni ALL, MERGE, DELETE).

COMMON lista di variabili

Utilizzata congiuntamente a CHAIN questa istruzione permette di passare solo alcune variabili al programma chiamato.

```
100     COMMON A,B,C,D(),G$
110     CHAIN "PIPPO",10
```

Anche se logicamente l'istruzione COMMON può essere posta in un punto qualsiasi del programma, in genere, per motivi di chiarezza, la si utilizza in testa al programma stesso.

Una variabile può essere utilizzata in un'unica istruzione di tipo COMMON.

Se il nome della variabile è seguito dai caratteri '()' si passa al nuovo programma l'intero vettore avente quel nome.

PEEK — POKE: (ACCESSO DIRETTO ALLA MEMORIA)

Le istruzioni PEEK e POKE permettono di accedere direttamente a una locazione di memoria per leggerne o modificarne il contenuto.

PEEK (indirizzo)

Questa funzione fornisce il contenuto (in forma decimale intera) della locazione di memoria specificata (l'indirizzo deve essere compreso tra 0 e 65536). Poichè il risultato rappresenta il contenuto binario di un byte, esso sarà un intero compreso tra 0 (configurazione binaria 00000000) e 255 (configurazione binaria 11111111).

```
100     A=PEEK 1000
110     PRINT HEX$(A)
120     PRINT HEX$(PEEK &H3E8)
```

POKE indirizzo, valore

Questa funzione inserisce nella locazione di memoria indicata la forma binaria del valore specificato che deve essere intero e compreso tra 0 e 255.

```
200 POKE 1000, 122
```

INP — OUT — WAIT (INPUT/OUTPUT)

INP (numero della porta)

Legge il contenuto di un byte sulla 'porta' specificata (il numero della porta deve

essere compreso tra 0 e 255).

A=INP (255)

OUT numero della porta, dato

Invia un byte nella 'porta' specificata. Entrambi i parametri di questa funzione possono essere forniti tramite delle espressioni.

200 OUT 32,50

WAIT numero della porta, maschera, selezione

Questa funzione sospende l'esecuzione del programma finchè un particolare bit sulla porta specificata non assume il valore 1.

– Se 'selezione' = 0 viene effettuato un AND tra il dato in entrata e 'maschera'. Quando il valore così ottenuto è diverso da zero il programma riprende l'esecuzione; in caso contrario invece viene nuovamente eseguita la funzione WAIT.

– Se 'selezione' # 0 viene prima effettuato un OR ESCLUSIVO tra il valore in entrata e 'selezione', in seguito viene effettuato un AND tra il risultato e 'maschera'. Si procede infine come nel caso precedente.

10 WAIT 22,4 Sospende l'esecuzione finchè il terzo bit ($4=2^2$) non è settato sulla porta 22

10 WAIT 22,7 sospende il programma finchè uno dei primi tre bit sulla porta 22 non risulta settato.

```
10'  STAMPA DI UNA MATRICE BIDIMENSIONALE
20'  ===================================================
30'
40   DATA "MILANO  ","TORINO","GENOVA","PALERMO","VENEZIA","NAPOLI"
              FOR I=1 TO 6:READ FILIALE$(I):NEXT I
50   DATA "R5","R6","R12","R18","R20","R30","R40"
              FOR I=1 TO 7:READ TIPO$(I):NEXT I
60   DIM STOCK(6,7)                 '7 TIPI E 6 FILIALI'
70'
80'                                 ACQUISIZIONE DATI
90   STOCK(1,1)=52:STOCK(1,2)=79:STOCK(1,3)=87:STOCK(2,2)=99:
     STOCK(2,5)=119
100'----------------------------------------------------------------
110'                                STAMPA CODICI FILIALI
120  FOR I=1 TO 5
130   LPRINT TAB(15+10)
140'                              VETTORE FILIALE$
150   FOR J=1 TO 6                ----------------
160     X$=MID$(FILIALE$(J),I,1):LPRINT X$;" ";
170   NEXT J
190   LPRINT
200  NEXT I
220  LPRINT
230'---------------------------------------------------------------
240'                                STAMPA DELLA MATRICE
250  FOR TIPO=1 TO 7
260     LPRINT TAB(15) TIPO$(TIPO);TAB(15+6)
270     TOTTIPO=0
290     FOR FILIALE=1 TO 6
300            IF STOCK(TIPO,FILIALE)<>0 THEN
                 LPRINT USING "#####";STOCK(TIPO,FILIALE);
               ELSE LPRINT"    .";
310            TOTTIPO=TOTTIPO+STOCK(TIPO,FILIALE)
320            MAGAZZ(FILIALE)=MAGAZZ(FILIALE)+STOCK(TIPO,FILIALE)
330     NEXT FILIALE
350     LPRINT USING"#####";TOTTIPO
360  NEXT TIPO
380  LPRINT:LPRINT TAB(15+6)
390'
400  FOR FILIALE=1 TO 6
410      LPRINT USING"#####";MAGAZZ(FILIALE);
420  NEXT FILIALE
430'
```

VETTORE FILIALE$

FILIALE$(1)	M	I	L	A	N
FILIALE$(2)	T	O	R	I	N
.	G	E	N	O	V
.	P	A	L	E	R
.	V	E	N	E	Z
FILIALE$(6)	N	A	P	O	L

← 5 →

	M I L A N	T O R I N	G E N O V	P A L E R	V E N E Z	N A P O L	
R5	52	79	87	.	.	.	218
R6	.	99	.	.	119	.	218
R12	.	.	.	.	.	.	0
R18	.	.	.	.	.	.	0
R20	.	.	.	.	.	.	0
R30	.	.	.	.	.	.	0
R40	.	.	.	.	.	.	0
	52	178	87	0	119	0	

```
20'          RICERCA DICOTOMICA
30'          ==================
40'
50'     IL METODO DICOTOMICO PER RINTRACCIARE UN CERTO ELEMENTO
60'     IN UNA TABELLA ORDINATA CONSISTE NEL SELEZIONARE AD
70'     OGNI PASSO UNA PORZIONE DI TABELLA SEMPRE PIU LIMITATA
80'     IN CUI IL NOSTRO ELEMENTO DEVE TROVARSI.
90'     IN PRATICA SI EFFETTUA UN CONTROLLO SULLO ELEMNTO CENTRALE
100'    DELLA TABELLA PER INDIVIDUARE LA META" IN CUI LO ELEMENTO
110'    CERCATO DEVE TROVARSI.
120'    SI PROCEDE POI NELLO STESSO MODO SULLA MEZZA TABELLA
130'    SELEZIONATA E COSI" VIA FINO A CHE O SI INCONTRA LO
140'    ELEMENTO CERCATO O, NEL CASO IN CUI ESSO NON SIA PRESENTE
150'    IN TABELLA, SI INDIVIDUA LA POSIZIONE IN CUI DEVE ESSERE
160'    INSERITO. (SE NELLA TABELLA VI SONO PIU" ELEMENTI
170'    UGUALI VIENE SELEZIONATO IL PRIMO).
180'
185'                                     +------+
190'       INFERIORE -------->           |  2   |
200'                                     +------+
210'    ELEMENTO CERCATO --->            |  4   |
215'                                     +------+
220'                                     |  6   |
225'                                     +------+
230'                                     |      |
240'                                     +------+
245'          CENTRALE------->           |  10  |
246'                                     +------+
250'                                     +------+
255'          SUPERIORE----->            |  20  |
256'                                     +------+
258'
260     LUNG=10            'GENERAZIONE DI UNA TABELLA DI 10 ELEMENTI
270     FOR I=1 TO LUNG:A(I)=2*I:NEXT I
280'
290     INPUT "VALORE CERCATO: ";V
300'
310     GOSUB 370:ON Q GOTO 320,330     'RICERCA DICOTOMICA'
320     PRINT"VALORE TROVATO NELLA POSIZIONE ";POS:GOTO 290
330     PRINT"VALORE INESISTENTE.PUOI INSERIRLO IN POSIZIONE ";POS
335     GOTO 290
350' ---------------------------------------------------------------
370     INFERIORE=1:SUPERIORE=LUNG
390     IF INFERIORE>SUPERIORE THEN Q=2:POS=CENTRALE:IF V>A(POS) THEN
           POS=POS+1:RETURN ELSE RETURN          'VALORE INESISTENTE
400     CENTRALE=INT((INFERIORE+SUPERIORE)/2)
420     IF V<>A(CENTRALE) THEN GOTO 440
425     IF A(CENTRALE-1)<>A(CENTRALE) THEN Q=1:POS=CENTRALE:RETURN
           ELSE CENTRALE=CENTRALE-1:GOTO 420
440     IF V<A(CENTRALE) THEN SUPERIORE=CENTRALE-1:GOTO 390
450     IF V>A(CENTRALE) THEN INFERIORE=CENTRALE+1:GOTO 390
460' ---------------------------------------------------------------

RUN

VALORE CERCATO 4
VALORE TROVATO NELLA POSIZIONE: 2
```

```
10'      RICERCA AUTOMATICA DEL FATTORE DI SCALA PER UN ISTOGRAMMA
20'      ==========================================================
30'
40' QUESTO PROGRAMMA PERMETTE DI COSTRUIRE UN ISTOGRAMMA SENZA
50' AVER DEFINITO A PRIORI IL FATTORE DI SCALA.
60'
70       DIM TX(20)
80'
90' GENERAZIONE DI 15 VALORI CASUALI PER PROVARE LA ROUTINE
100'
110      FOR I=1 TO 15:TX(I)=INT(RND(1)*20:LPRINT USING "###";TX(I):
         NEXT I
120'-------------------------------------------------------------------
130      FOR I=1 TO 15                        'RICERCA DEL MASSIMO'
140          IF TX(I)>MAX THEN MAX=TX(I)
150      NEXT I
160'
170      SCALA=10/MAX                         'ALTEZZA MASSIMA 10'
180      SCALA=INT(SCALA*8)/8                 'FATTORE MINIMO DI SCALA 0.125'
190'-------------------------------------------------------------------
200'                                         'STAMPA'
210      LPRINT:LPRINT:LPRINT"ISTOGRAMMA":LPRINT""
220      FOR J=10 TO 1 STEP -1
230                FOR I= 1 TO 15
240                    IF TX(I)*SCALA>=J THEN LPRINT"  *";
                                          ELSE LPRINT"   ";
250                NEXT I
260                LPRINT""
270      NEXT J
280'
290      LPRINT"":FOR I=1 TO 15:LPRINT USING "###";I;:NEXT I
300      LPRINT"     SCALA: ";SCALA
310      END
```

```
   4  6  6 10  1 15  9  7 19 18 14  0 19  0 19


ISTOGRAMMA
```

```
10 '      ORDINAMENTO CON IL METODO DI SHELL
20 '      ==================================
30 '
40 '      IL CONFRONTO NON SI EFFETTUA TRA DUE VALORI CONSECUTIVI,
50 '      COME PER IL METODO <RIPPLE> MA TRA DUE VALORI SEPARATI
60 '      DA UN "PASSO" CHE ALL"INIZIO E" UGUALE ALLA META" DELLA
70 '      DIMENSIONE DELLA TABELLA DA ORDINARE.
80 '      . AL SECONDO GIRO IL PASSO VIENE DIVISO PER 2 E COSI VIA
90 '      . L"ULTIMO PASSAGGIO VIENE EFFETTUATO CON PASSO 1 (COME
100'        PER IL METODO RIPPLE)
110'
120'
130       INPUT "DIMENSIONE: ";D
140       DIM A(D)
150'
160       FOR I=1 TO D: A(I)=RND(1): NEXT I   'GENERAZIONE TABELLA'
170'
180       GOSUB 280: FOR I=1 TO D: PRINT A(I): NEXT I
190       PRINT "DIMENSIONE ";D;TAB(20) "CONFRONTI ";C;
                TAB(40) "SCAMBI ";S
200       END
210 '     -----------------------------------------------------------
220 '
230 '     9 8 7 6 5 4 3 2 1 0
240 '     -         -
250 '     !         !
260 '     INVERSIONE
270 '
280       PAS=D
290 '
300       PAS=INT(PAS/2):IF PAS<1 THEN RETURN
310       INVERSIONI=0
320 '
330       FOR I=1 TO D-PAS
340          C=C+1          'CONTATORE DEI CONFRONTI'
350          N=I+PAS: IF A(I)>A(N) THEN
               SWAP A(I),A(N):INVERSIONI=1:S=S+1
360       NEXT I
370 '
380       IF INVERSIONE=1 THEN GOTO 310 ELSE GOTO 300
390 '     SI RICOMINCIA FINCHE" NON VI SONO ZERO INVERSIONI
400 '
410 '     ------------------------------------------------------
420 '
430 '                          SHELL                         RIPPLE
440 '
450 '     DIMENSIONE : CONTRONTI  INVERSIONI    : CONFRONTI  INVERSIONI
460 '
470 '          10    :     47          8        :       45         19
480 '          20    :    230         39        :      190        104
490 '          50    :   1010        180        :     1170        642
500 '         100    :   3382        515        :     4914       2657
510 '         200    :   6791       1035        :    19669      10541
520 '
530 '     -------------------------------------------------------------
```

CAPITOLO 2

FILES AD ACCESSO DIRETTO (random)

I files permettono di memorizzare in modo permanente delle informazioni che così possono essere utilizzate anche dopo parecchio tempo. I supporti fisici che si utilizzano per la memorizzazione dei files sono in genere nastri o dischi magnetici.

Faremo in particolare riferimento ai floppy-disks, cioè a quei particolari dischi magnetici utilizzati sulla maggior parte dei Personal Computers.

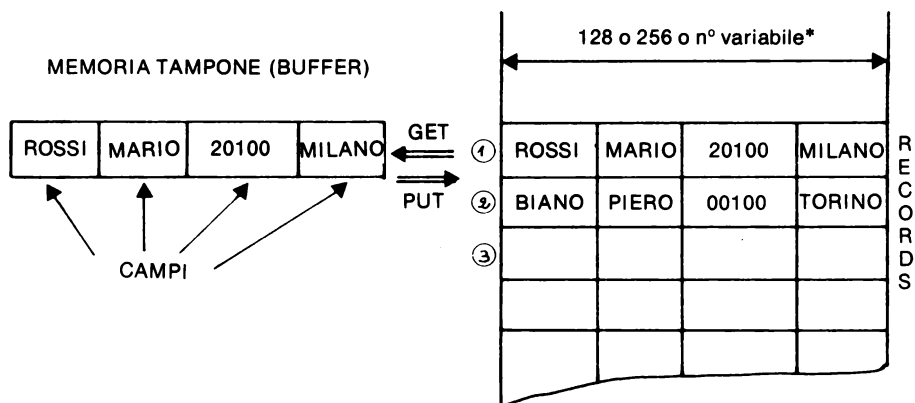
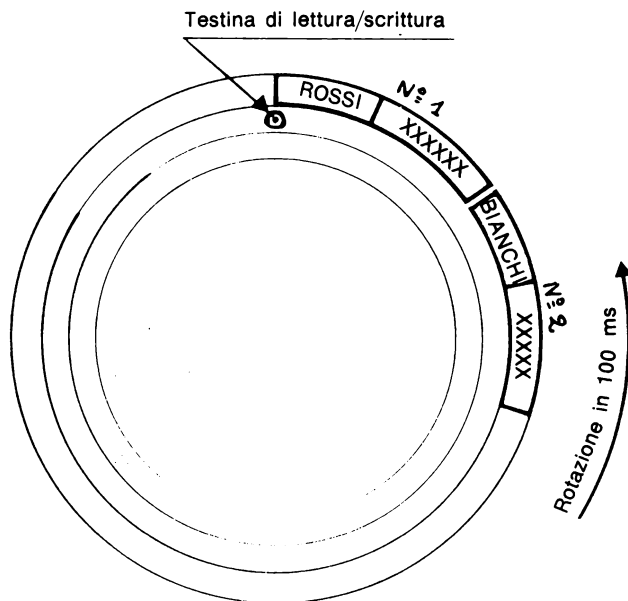
Un floppy-disk è composto da più tracce concentriche (35 per i dischi da 5 pollici e mezzo e 77 per quelli da 8 pollici) e ogni traccia è a sua volta suddivisa in "settori" (rispettivamente 10 e 26). Ciascun settore (che può contenere o 128 o 256 caratteri) è individuato sul disco da un 'indirizzo fisico' (numero della traccia e posizione del settore all'interno della traccia stessa).

L'accesso ad un settore viene ottenuto nel modo seguente: dapprima la testina di lettura-scrittura si posiziona sulla traccia selezionata e poi, mediante una ricerca sulla traccia, all'inizio del settore selezionato.

Nonostante in realtà la ricerca all'interno della traccia sia di tipo sequenziale, quando è possibile selezionare direttamente un settore si parla di organizzazione ad accesso diretto. Ciò è dovuto al fatto che la ricerca del record richiesto appare all'utente effettivamente di tipo diretto in quanto la richiesta di accesso viene fatta utilizzando direttamente l'indice di posizione del record nel file.

In realtà però è il sistema operativo che ricava da questa informazione l'indirizzo fisico del settore in cui è memorizzato il record ed effettua tramite questo una ricerca sequenziale sulla traccia selezionata.

Le vecchie versioni del BASIC permettevano di utilizzare solo records di lunghezza prefissata (128 o 256 caratteri). Sulle versioni più moderne è invece il programmatore, tramite la definizione dei campi, a decidere la lunghezza del record. È meglio ricordare comunque che anche in questo caso tutti i records di uno stesso file devono avere la stessa lunghezza.

Utilizzando i files ad accesso diretto è possibile ottenere un aggiornamento di un fi-

PUT = SCRITTURA
GET = LETTURA

* 128 bytes	X1; DTC MICROFILE
* 256 bytes	TRS80 NEW DOS
* lunghezza variabile	MICROSOFT 5.

FILES AD ACCESSO DIRETTO

le in modo immediato (in tempo reale) mentre ciò non era possibile con i files sequenziali.

OPEN

Per poter eseguire un'operazione di lettura o scrittura su un file è necessario che esso sia 'aperto' a tale operazione.

L'istruzione OPEN, che 'apre' un file per la lettura o la scrittura, fa sì che venga predisposta in memoria un'area tampone, detta buffer, nella quale verranno via via depositati i vari dati che vengono trasferiti dal file (lettura) o che dovranno esservi trasferiti (scrittura).

OPEN "R", *#n° file, "U:nome file", lunghezza record*
←——— MICROSOFT 5. (U rappresenta l'unità disco su cui risiede il file: U=A, B, C ... Se U non è specificato viene considerato come A)

OPEN "R", *#n° file, "nome file: n° unità disco"*
←——— TRS80 NEW DOS (Se l'unità disco non viene specificata vien considerato il disco 0)

"R" specifica il tipo di file (RANDOM). Questo parametro può essere definito anche tramite una variabile o un'espressione.

'n° file' anche se viene utilizzato praticamente come identificatore del file, in realtà rappresenta il numero del buffer assegnato al file stesso durante l'operazione di 'apertura'.

'U: nome file' può essere specificato anche tramite un'espressione (ad esempio "A:"+NOMFIL$).

'lunghezza record', che può essere specificata tramite un'espressione, è posta automaticamente pari a 128 (Microsoft 5.) qualora questo parametro non venga utilizzato nell'istruzione OPEN.

```
A    10 OPEN "R",#1,"CLIENTI",200

B    10 INPUT "NOME FILE";NF$
     20 OPEN "R",#1,NF$,64
```

Nell'esempio A l'istruzione OPEN apre il file CLIENTI assegnandogli il buffer n° 1

(nel seguito del programma il file verrà indicato come #1) e considera ogni record formato da 200 caratteri.

Nella versione Microsoft 5. vengono definiti in fase di inizializzazione del sistema sia il numero massimo di files che possono venire aperti contemporaneamente, sia la lunghezza massima ammessa per un record.

MBASIC/S: *lunghezza max record*/F: *numero max files*

Uno stesso file può essere aperto con diversi numeri. In tal caso al file stesso vengono associati più buffers (cfr aperture multiple).

FIELD# n° file, l1 AS var1$, l2 AS var2$,...
(DESCRIZIONE DEI CAMPI)

Un record è identificato all'interno di un file ad accesso diretto da un numero che rappresenta la sua posizione nel file stesso.* Ciascun record è a sua volta suddiviso in 'campi', la cui lunghezza viene specificata mediante l'istruzione FIELD.

FIELD# *n° file, l1* AS *var1$, l2* AS *var2$*, ...

Questa istruzione, che descrive la struttura dei record specificando sia la posizione dei vari campi all'interno del record, sia la loro lunghezza, viene in genere utilizzata congiuntamente all'istruzione OPEN.

20 FIELD#1, 15 AS COGNOME$, 10 AS NOME$, 25 AS VIA$, 5 AS CAP$

15 caratteri sono riservati per il cognome
10 caratteri sono riservati per il nome
25 caratteri sono riservati per l'indirizzo
5 caratteri sono riservati per il codice postale

Durante le operazioni di lettura e scrittura si può accedere ad *un solo record per volta* (un solo record può essere trasferito nel buffer).

PUT# n° file, n° record (per scrittura) **LSET** **RSET**

Per poter effettuare l'operazione di scrittura in un record di un file è necessario de-

positare prima nei vari campi definiti nel buffer i dati che dovranno poi essere trasferiti nel file.

Per effettuare questa operazione preliminare si utilizza in genere una delle due funzioni LSET o RSET.

LSET *nome campo = stringa di caratteri (o espressione)*

```
10        INPUT"COGNOME: ";X$
20        LSET COGNOME$=X$      'TRASFERISCE X$ NEL BUFFER
25'                              (CAMPO  COGNOME$)
30        INPUT"NOME: ";X$
40        LSET NOME$=X$         'TRASFERISCE X$ NEL BUFFER
45'                              (CAMPO NOME$)
               :
               :
               :
100       PUT #1,10             'TRASFERISCE IL CONTENUTO DEL
101'                             BUFFER NEL DECIMO RECORD DEL FILE
```

Con l'istruzione LSET, il dato viene caricato nel campo del buffer occupando nel campo stesso le prime posizioni a sinistra. (Utilizzando invece RSET si sarebbe avuto un caricamento a destra). I bytes del campo rimasti inutilizzati vengono completati con degli spazi bianchi (codice ASCII 32). Ad esempio LSET COGNOME$="ROSSI" inserisce i 5 caratteri R, O, S, S,I, nelle prime 5 posizioni a sinistra del campo COGNOME$ e completa il campo stesso con 10 spazi bianchi (se il campo è stato definito di 15 caratteri).

Attenzione: le variabili utilizzate per definire i campi all'interno di un buffer non possono venire inizializzate direttamente con dei dati provenienti dall'esterno. Pertanto non sarà ammessa la forma INPUT NOME$ se NOME$ è il nome di un campo di un buffer.

Anche se, in fase di aggiornamento di un record, si vogliono modificare solo alcuni campi è indispensabile aggiornare anche i rimanenti campi nel buffer prima di effettuare l'istruzione PUT. In caso contrario infatti verrebbero considerati, per i campi non aggiornati, i valori già presenti nel buffer e derivanti da una precedente operazione di lettura/scrittura.

Questo inconveniente può essere evitato, **se si utilizzano records di lunghezza fissa (128 o 256 bytes),** nel modo seguente **(cfr LOF e LOC per il Microsoft 5.)**: prima di effettuare il trasferimento dei dati nel buffer, si legge dal file un record inesistente

GET#1, LOF(1)+1

in modo da avere nel buffer una sequenza di zeri binari.

```
5        GET #1.LOF(1)+1        'INIZIALIZZAZIONE DEL BUFFER
10       INPUT"NOME: ";X$
20       LSET NOME$=X$
                :
                :
                :
100      PUT #1.10              'SCRITTURA SUL FILE DEL RECORD 10
```

Attenzione: sul TRS80, generando il record 100 in un file vuoto, si ottiene la generazione automatica dei precedenti 99 records assegnando ad essi dei valori casuali.

Numeri di record

MICROSOFT 5.	da 1 a 32567
TRS80	da 1 a 355
DTC MICROFILE	da 0 a 4095

Se nell'istruzione PUT non viene specificato il numero del record a cui ci si riferisce, la scrittura viene effettuata nel record successivo a quello corrente.

GET# n° file, n° record **(LETTURA)**

La lettura di un record (trasferimento di dati dal disco al buffer) viene effettuata tramite l'istruzione:

GET#*n° file, n° record*

Dopo l'esecuzione di questa istruzione sono disponibili nel buffer i valori relativi a tutti i campi del record; pertanto si può richiedere, ad esempio, la loro visualizzazione mediante un PRINT.

100 PRINT COGNOME$,NOME$,VIA$
con COGNOME$,NOME$,VIA$ descrittori dei campi del record

Come per l'istruzione PUT, anche nel caso di GET se non si specifica il numero del record si intende far riferimento al record successivo a quello corrente.

AGGIORNAMENTO DEI CAMPI

Volendo modificare il contenuto di un campo di un certo record si devono eseguire le tre seguenti operazioni:

. lettura (GET) del record interessato
. variazione del campo nel buffer
. scrittura del record modificato (PUT).

```
10      GET #1,X                        'LETTURA DEL RECORD X
20      LSET CITTA$="MILANO"            'MODIFICA DI UN CAMPO
30      PUT #1,X                        'RISCRITTURA SUL FILE
```

CAMPI DI TIPO NUMERICO: MKI$ MKS$ MKD$ CVI CVS CVD

Poichè un campo può essere definito soltanto tramite una variabile di tipo alfanumerico, se si vuole assegnare ad un campo un valore numerico occorre prima trasformare tale valore in una stringa di caratteri.

Questo tipo di trasformazione viene effettuato utilizzando la funzione MKx$ nel modo seguente:

LSET *campo* = MKx$(espressione numerica)

ove x è un carattere dipendente dal tipo del valore numerico che si vuole trasformare, e precisamente:

x = I per valori di tipo intero
x = S per valori reali in semplice precisione
x = D per valori reali in doppia precisione

```
10      INPUT"CODICE POSTALE";CAP
20      LSET CPPST$=MKS$(CAP)
30      PUT #1,X
```

Successivamente, in fase di lettura, la conversione inversa verrà effettuata mediante la funzione CVx come illustrato nell'esempio seguente:

```
50      GET #1,X
60      CAP=CVS(CPPST$)
70      PRINT CAP
```

La tabella seguente illustra, dipendentemente dal valore numerico, la funzione utilizzata per la conversione e lo spazio occupato

0 ——→ 255	LSET X$=CHR$(X)	1 carattere	X=ASC(X$)
−32768 ——→ +32767	LSET X$=MKI$(X)	2 caratteri	X=CVI(X$)
semplice precisione	LSET X$=MKS$(X)	4 caratteri	X=CVS(X$)
doppia precisione	LSET X$=MKD$(X)	8 caratteri	X=CVD(X$)
	scrittura	bytes occupati	lettura

Definizione di strutture multiple

Si possono definire contemporaneamente più strutture per uno stesso record.

```
10        FIELD #1,15 AS COGNOME$,10 AS NOME$,.......
20        FIELD #1,15 AS CAMPO$(1),10 AS CAMPO$(2),.......
30        FIELD #1,25 AS SIG$,.......
```

Nell'esempio precedente il campo NOME viene riconosciuto sia come NOME$ che come CAMPO$(2). Con questo accorgimento si può eccedere ad un campo sia mediante un nome simbolico (NOME$) sia specificando la sua posizione nel record (CAMPO$(2)).

La terza istruzione ristruttura i due campi COGNOME$ e NOME$ unificandoli in un unico campo denominato SIG$.

Questo avviene in quanto praticamente l'istruzione FIELD inizializza solo i puntatori alla testa dei vari campi.
Assegnando più 'nomi' allo stesso campo si ottiene pertanto l'inizializzazione di più puntatori sullo stesso indirizzo.

Nella versione 5. un'istruzione FIELD relativa ad un file non aperto permette di definire una struttura su una riga di caratteri.

Forma compattata

Se tutti i campi di un record hanno la medesima lunghezza la definizione dei campi stessi può essere fatta come nell'esempio seguente:

Ogni campo contiene cinque caratteri

```
10        FOR I=1 TO 10
20                  FIELD #1,(I-1)*5 AS D$,5 AS CAMPO$(I)
30        NEXT I
```

La variabile D$ è utilizzata per posizionare correttamente i puntatori relativi ai vari campi.

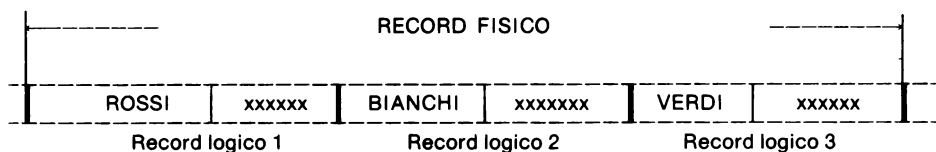
Definizione di più tabelle

Nell'esempio vengono memorizzati in un file due vettori A$ e B$.

3	2	3	2	
A$(1)	B$(1)	A$(2)	B$(2)	....

```
10       FOR I=1 TO 10
20                 FIELD #1,(I-1)*5 AS D$,3 AS A$(I),2 AS B$(I)
30       NEXT I
```

Records logici

Quando non è possibile fissare arbitrariamente le lunghezze dei records si può ricorrere, per memorizzare files formati da records piuttosto corti, alla scomposizione di un record fisico in più records logici.

Detto NLOG il numero di record logico NFIS quello del record fisico a cui appartiene e POS la posizione del record logico all'interno di quello fisico, sussistono le seguenti relazioni:

NFIS = (INT(NLOG − 1)/3) + 1 (ogni record fisico contiene 3 records logici)

POS = ((NLOG − 1) MOD 3) + 1

Esempio: NLOG = 8 ——→ NFIS = INT ((8 − 1)/3) + 1 = 3
NLOG = 8 ——→ POS = ((8 − 1) MOD 3) + 1 = 2

La definizione dei campi all'interno di un record logico può essere fatta dinamicamente come illustrato nell'esempio seguente:

```
10      INPUT"NUMERO RECORD LOGICO ";NLOG
20      NFIS=INT((NLOG-1)/3)+1
30      POS=((NLOG-1) MOD 3)+1:GOSUB 1000
40      GET #1,NFIS
50      LSET COGNOME$="ROSSI"
60      PUT #1,NFIS

1000    FIELD #1,(POS-1)*50 AS D$,15 AS COGNOME$,....:RETURN
```

Poichè ogni variabile deve essere assegnata ad uno dei 3 records logici, l'istruzione FIELD deve essere eseguita ogni volta che si cambia record logico.

Attenzione: L'esecuzione dell'istruzione FIELD risulta piuttosto lunga se il numero dei campi da definire è elevato.

Per questo motivo ci sembra opportuno presentare una soluzione alternativa alla precedente, utilizzabile però solo nel caso in cui ogni record fisico sia diviso in *due* records logici. In questo caso, se si vuole operare sul secondo record logico, dopo la lettura dell'intero record fisico si salva in una variabile il contenuto del primo record

logico, si trasferisce la seconda metà del buffer (2° record logico) nella prima (in cui mediante un FIELD sono state definite le variabili), si effettua l'aggiornamento del record logico, si ritrasferisce la prima parte del buffer nella seconda e si ricopia nella prima il record precedentemente salvato. In tal modo l'intero record fisico viene ristrutturato per la scrittura.

Questa procedura è illustrata nell'esempio seguente:

```
50      FIELD #1,15 AS COGNOME$, 10 AS NOME$,.......
60      FIELD #1,128 AS E1$,128 AS E2$        'DEFINIZIONE DI 2 RECORDS
                                                     LOGICI'
:         :              :         :

170     GET #1,NFIS
180     IF POS=2 THEN REC1$=E1$:LSET E1$=E2$
190     LSET COGNOME$="ROSSI"
200     IF POS=2 THEN LSET E2$=E1$:LSET E1$=REC1$
210     PUT #1,NFIS
```

Aperture multiple per un file

È possibile, mediante più istruzioni OPEN, assegnare ad uno stesso file più buffers (attribuendogli così più numeri di identificazione). Con questo accorgimento è possibile avere contemporaneamente in memoria più records dello stesso file.

Questa opzione, peraltro non ammessa da tutti i Basic, va comunque usata con molta prudenza. In particolare converrà utilizzare un numero di file per la CREAZIONE di nuovi records ed un altro per la manipolazione dei records già esistenti.

In questo caso il secondo buffer non verrà mai chiuso (il descrittore dinamico non necessita infatti di aggiornamento).

Nella versione 5. utilizzando le aperture multiple un file può essere aperto sia ad accesso sequenziale che ad accesso diretto.

CLOSE # n° file, # n° file, ...

Questa istruzione serve per 'chiudere' un file e rilasciare il buffer che gli era stato assegnato.

Quando un file viene aperto, viene trasferito in memoria centrale un "descrittore dinamico" del file stesso che contiene dei puntatori ai vari records del file. Questo descrittore viene aggiornato, in memoria centrale, ogni qual volta si rende necessaria la creazione di nuovi records.

Su diversi sistemi (tra cui il TRS 80) l'aggiornamento del descrittore dinamico su disco avviene solo in concomitanza con la chiusura del file. Pertanto, se ci si dimenti-

ca di chiudere dei files prima di porre termine al lavoro, si perdono tutti i descrittori relativi a tutti i records aggiunti durante quella fase.

Per evitare questo inconveniente si può ricorrere ad un espediente che però annulla tutti i vantaggi della allocazione dinamica. Tale espediente consiste nel generare all'inizio del programma un record vuoto con un numero di identificazione molto alto e nel chiudere poi il file. In questo modo vengono memorizzati nel descrittore dinamico tutti i puntatori ed i records che dovrebbero precedere quello generato e pertanto, in genere, non è più necessario aggiornare di nuovo il descrittore stesso.

CLOSE #1, #2

I comandi END e NEW provocano automaticamente la chiusura di tutti i files.

LOF LOC (PER RECORDS DI LUNGHEZZA FISSA PARI A 128 O 256)

LOF (n° file)

Permette di determinare il numero di records presenti in un file e quindi la posizione in cui dovrà essere aggiunto un nuovo record.

```
100     NUOVO=LOF(1)+1              'POSIZIONE NUOVO RECORD'
120     LSET C$(1)="PIPPO"          'SCRITTURA NEL BUFFER'
130     LSET C$(2)="PLUTO"          'SCRITTURA NEL BUFFER'
150     PUT #1,NUOVO                'MEMORIZZAZIONE NEL FILE'
```

Su alcuni sistemi, come il DTC Microfile, la funzione LOF fornisce direttamente la posizione del nuovo record (LOF = lunghezza file + 1).

LOC (n° file)

Permette di posizionarsi sul record successivo a quello utilizzato nell'ultima operazione di input-output.

Nota: le operazioni di lettura e scrittura in un file possono essere effettuate anche utilizzando direttamente dei comandi da consolle. Questa opzione viene usata con grande utilità in fase di messa a punto.

```
GET# 1,3: PRINT N$
PIPPO
OK

LSET N$="XXXX": PUT # 1,3
OK
```

LOF (n° file) (microsoft 5.)

Nella versione 5. la funzione LOF restituisce il numero di blocchi di 128 bytes utilizzati nell'extent (1 extent = 128*128 bytes) a cui si è fatto riferimento nell'ultima operazione di input-output. Nel computo vengono conteggiati per intero anche eventuali blocchi utilizzati solo parzialmente.

Con la versione Microsoft 5. pertanto la funzione LOF restituisce il numero di records presenti nel file solo nel caso in cui i records siano di lunghezza fissa pari a 128 bytes e il loro numero non superi 128.

Come si può fare allora in genere a rintracciare la posizione in cui è possibile aggiungere un nuovo record?

- La soluzione più semplice consiste nell'utilizzare un contatore che contenga sempre il numero di records presenti nel file. Tale contatore potrà essere memorizzato ad esempio nel primo record.

- Una soluzione alternativa consiste nell'utilizzare una BIT-MAP che descriva l'occupazione di memoria di massa per il file (cfr allocazione dinamica).

LOF (n° file) (TRS80)

Su questo sistema la funzione LOF fornisce il numero di blocchi di 256 bytes utilizzati dal file. In questo caso il valore restituito coincide con il numero di records solo se questi hanno lunghezza pari a 256 caratteri.

EOF (n° file)

(5.) Restituisce il valore −1 se in fase di lettura si cerca di far riferimento ad un record che dovrebbe trovarsi al di fuori dello spazio assegnato al file.

Errori classici

Vengono qui di seguito segnalati degli 'errori' abbastanza frequenti e molto pericolosi in quanto non riconosciuti come tali dal sistema.

- La mancanza di un LSET nell'inizializzazione del buffer prima di un PUT. A causa di questo errore può capitare di non ritrovare più nel record ciò che si pensava di aver scritto.
  (Per controllare ciò che si è realmente scritto in un record si può ricorrere al seguente espediente: rileggere immediatamente dopo la scrittura il record interessa-

to (GET) e visualizzare il suo contenuto. Eseguendo questa operazione direttamente da consolle è possibile modificare direttamente il contenuto del record nel caso in cui si sia riscontrata un'inesattezza).

- L'uso in un'istruzione INPUT di una variabile precedentemente utilizzata come descrittore di campo. A causa di qusto errore la variabile stessa perde la sua funzione di descrittore.

- L'uso della stessa variabile in più istruzioni FIELD. In questo caso il sistema assegna la variabile solo al buffer specificato nell'ultima istruzione FIELD.

- Sovrapposizione (non voluta) di più campi. Questo errore è generato dall'uso non corretto della dichiarazione multipla per i campi.

- Conversione errata di dati numerici in fase di lettura mediante le funzioni CVx. È evidente infatti che per una corretta interpretazione delle informazioni, un dato convertito in stringa mediante un MKI$ (p. es.) deve poi essere riconvertito mediante la funzione corrispondente CVI.

- Non effettuare un GET preventivo se si deve modificare solo qualche campo, del record. In questo caso, come abbiamo visto, viene falsata completamente l'operazione di aggiornamento del record stesso.

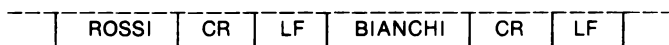
```
10        OPEN "R",#1,"XX",50
20        FIELD #1,...DESCRITTORI DEI CAMPI....' TOTALE 50 CARATTERI'
30        FIELD #1,2 AS NREC$,48 AS REC$
40        GET #1,1:CONTATORE%=CVI(NREC$)

ACQUISIZIONE DATI NUOVO RECORD

100       CONTATORE%=CONTATORE%+1
110       PUT #1,CONTATRE%
120       LSET NREC$=MKI$(CONTATORE%)
130       PUT #1,1                  'MEMORIZZAZIONE NUOVO VALORE DEL
                                     CONTATORE
```

```
20'  .    FILES AD ACCESSO DIRETTO (RANDOM)
30'       ==================================
40'  **** GENERAZIONE,CONSULTAZIONE,AGGIORNAMENTO ****
50'
60'       I FILES PERMETTONO DI CONSERVARE DELLE INFORMAZIONI
70'       (I VETTORI INVECE SONO DELLE MEMORIE DI LAVORO)
80'
120'                 ----------------------------------
130'         1       ! ROSSI   ! MARIO ! 02-651548 !   <--- RECORD 1
140'                 ----------------------------------
150'         2       !         !       !           !
160'                 ----------------------------------
170'         3       !         !       !           !
180'                 ----------------------------------
190'                  CGN$       N$       TEL$        <--- CAMPI
200'
210'                                 GENERAZIONE DI UN FILE
220'                                 ======================
230'
240 REM              1) APERTURA DEL FILE (OPEN)
250 REM              2) DESCRIZIONE DEI CAMPI DI UN RECORD (FIELD)
260 REM              3) SCRITTURA (PUT#)
270'
280       OPEN "R",#1,"CLIENTI",40  'APRE IL FILE CLIENTI ASSEGNANDOGLI
                                     IL BUFFER NUMERO 1'
290       FIELD #1,10 AS CGN$,10 AS N$,20 AS TEL$ 'DEFINIZIONE CAMPI'
300'
320       PRINT:PRINT "FUNZIONE(GEN,CONS,AGG,LIST,KEY,FINE) ";F$
325'
330       IF F$="GEN" THEN GOSUB 420     'GENERAZIONE NUOVO RECORD'
340       IF F$="KEY" THEN GOSUB 830     'RICERCA NOMINATIVA'
350       IF F$="LIST" THEN GOSUB 960    'STAMPA DI TUTTO IL FILE'
360       IF F$="CONS" THEN GOSUB 590    'CONSULTAZIONE RECORD'
370       IF F$="AGG" THEN GOSUB 700     'AGGIORNAMENTO RECORD'
380       IF F$="FINE" THEN STOP
390       GOTO 320
400'
410' -----------------------------------------------------------------
415'                                       GENERAZIONE DEL FILE
420'                                       ====================
430       NREC=LOF(1):GET #1,NREC    'AGGIUNTA DEL RECORD'
440       PRINT:INPUT"COGNOME NUOVO CLIENTE ";CLIENTE$
450       IF CLIENTE$="" THEN 560
460       LSET CGN$=CLIENTE$
470       INPUT"NOME ";NOME$
480       LSET N$=NOME$
490       INPUT"TELEFONO";TH$
500       LSET TEL$=TH$
505'
510       PUT #1,NREC                'SCRITTURA NEL FILE'
520       GOTO 430
560       RETURN
580' ------------------------------------------------------------------
590       PRINT:INPUT"N. RECORD DA CONSULTARE ";NREC  'CONSULTAZIONE'
600       IF NREC=0 THEN 640                          '============='
610       GET #1,NREC
620       PRINT:PRINT CGN$,N$,TEL$:PRINT
```

```
630        GOTO 590
640        RETURN
650'  ------------------------------------------------------------------------


700        PRINT:INPUT"N. RECORD DA MODIFICARE ";NREC   'AGGIORNAMENTO'
705        IF NREC=0 THEN 790
710        GET #1,NREC
720        PRINT
730        PRINT USING "\           \";CGN$;:INPUT VAR$:IF VAR$<>"" THEN
                LSET CGN$=VAR$
740        PRINT USING "\           \";N$;:INPUT VAR$:IF VAR$<>"" THEN
                LSET N$=VAR$
750        PRINT USING "\                      \";TEL$;
760        INPUT VAR$:IF VAR$<>"" THEN LSET TEL$=VAR$
770        PUT #1,NREC                'MEMORIZZAZIONE NUOVI DATI'
780        GOTO 700
790        RETURN
800'  ------------------------------------------------------------------------
810                                   'RICERCA PER NOME'
820                                   '================'
830        PRINT:INPUT"COGNOME CLIENTE";X$:IF X$="" THEN 930
835        FF=LOF(1)-1                            'FF=FINE FILE'
836'
840        FOR I=1 TO FF                          'RICERCA SEQUENZIALE'
850                GET #1,I
860                IF X$=LEFT$(CGN$,LEN(X$)) THEN I=FF: T$="YES"
870        NEXT I
880 '
890        PRINT:IF LEFT$(T$,1)<>"Y" THEN PRINT "SCONOSCIUTO":GOTO 830
900'
910        PRINT CGN$,N$,TEL$          'STAMPA RECORD DESIDERATO'
920        T$="XXX":GOTO 830
930        RETURN
940'  ------------------------------------------------------------------------
950                                                'STAMPA FILE'
960        PRINT:PRINT"    ELENCO CLIENTI":PRINT   '==========='
970'
980        FOR I=1 TO LOF(1)-1
990                GET #1,I
1000               IF ASC(CGN$)=0 THEN GOTO 1020 'RECORD VUOTO'
1010               PRINT:PRINT CGN$,N$,TEL$
1020       NEXT I
1030'
1040 .     RETURN
1050'  -----------------------------------------------------------------------
1060'
1070'
1080'              .CON MICROSOFT 5. ATTENZIONE:
1090'               NELL"USO DELLA FUNZIONE LOF
1100'
1110'              .CON TRS80 NEW DOS
1120'                 CLEAR 3000
1130'                 CAMBIARE LOF(1) IN LOF(1)+1
1140'                 PRIMA DI UN INPUT X$ PORRE   X$=""

1150'
1160'               RICORDARSI DI CHIUDERE I FILES !!!!!!!!!!!!
1170'
1180'  ---------------------------------------------------------------------
```

CAPITOLO 3

FILES SEQUENZIALI

Nei files di tipo sequenziale l'aggiunta di nuovi elementi può essere fatta solo alla fine del file, mentre l'operazione di lettura può avvenire solo partendo dall'inizio del file stesso. Pertanto per raggiungere una certa posizione nel file occorre prima leggere tutte le informazioni che sono contenute nelle posizioni precedenti. Le operazioni di lettura e scrittura sono rette dalla medesime istruzioni (INPUT PRINT) utilizzate per l'acquisizione/visualizzazione di dayi da/su un videoterminale.

In generale i RECORDS sono separati tra loro da due caratteri speciali (detti elementi separatori): Carriage return (CR) e Line feed (LF) i cui codici ASCII sono rispettivamente 13 e 10.

OPEN

Sul sistema Microsoft 5. un file di tipo sequenziale è aperto ad un solo tipo di operazione per volta (lettura o scrittura). La forma dell'istruzione OPEN in questo caso sarà la seguente:

OPEN "0",# n° file, "U:nome file" (per la scrittura)
OPEN "I", # n° file, "U:nome file" (per la lettura)

O ——————→ output
I ——————→ input

10 OPEN "0",# 1,"A:PIPPO" ——————→ MICROSOFT 5.
10 OPEN "0",# 1,"PIPPO:1" ——————→ TRS80 NEWDOS

Attenzione: Il sistema Microsoft 5. al momento dell'apertura di un file per OUTPUT lo inizializza a zero!

PRINT# INPUT#

Il modo più semplice per scrivere in un file di tipo sequenziale è fornito dalla forma:

PRINT# n° file, variabile

Se si utilizza il programma:

```
90      INPUT "NOME,TELEFONO ";N$,TEL$
130     PRINT #1,N$
140     PRINT #1,TEL$
```

viene realizzata l'aggiunta di due records alla fine del file 1.
In modo analogo per leggere due records dal file si userà l'istruzione

INPUT# n° file, variabile

come illustrato nell'esempio.

```
220     INPUT #1,NOME$
230     INPUT #1,TEL$
250     PRINT NOME$,TEL$
```

In un'unica istruzione INPUT possono comparire più variabili, purchè separate da virgole.

30 INPUT# 1,NOME$,TEL$

Anche in questo caso comunque l'assegnazione alle variabili dei valori contenuti nel file viene fatta in modo sequenziale.

```
10'                    I FILES SEQUENZIALI
20'                    ===================
40'
50'                                                  SRITTURA
60'                                                  --------
70'
80    OPEN "O",#1,"CLIENTI"                          'APERTURA FILE CLIENTI
90    INPUT "NOME  ";NOM$                            'PER OUTPUT CON BUFFER
100   IF NOM$="" THEN CLOSE #1:GOTO 195              'NUMERO 1
110   INPUT "TELEFONO ";TEL$
120'
130   PRINT #1,NOM$                                  '<---SCRITTURA DI 1 RECORD
140   PRINT #1,TEL$                                  '<---SCRITTURA DI 1 RECORD
160   GOTO 90
170'-----------------------------------------------------------------------
180'                                                  I LETTURA
190'                                                  ---------
195   LPRINT "PRIMA PASSATA":LPRINT
200   OPEN "I",#1,"CLIENTI"                          'APERTURA DEL FILE CLIENTI
```

```
210'                                             PER INPUT CON NUMERO 1
220     INPUT #1,NOM$                            '<---LETTURA DI 1 RECORD
230     INPUT #1,TEL$                            '<---LETTURA DI 1 RECORD
240'
250     LPRINT NOM$,TEL$
260     IF EOF(1)=-1 THEN CLOSE #1:LPRINT:LPRINT:GOTO 315 'FINE FILE'
270     GOTO 220                                 'POSIZIONAMENTO SUL
275'                                             RECORD SUCCESSIVO
280' ---------------------------------------------------------------------
290'                                             II  LETTURA
300'                                             -----------
310     LPRINT "SECONDA PASSATA":LPRINT
320     OPEN "I",#1,"CLIENTI"
330'
340     INPUT #1,NOM$,TEL$                       '<---LETTURA DI 2 RECORDS
350'
360     LPRINT NOM$,TEL$
370     IF EOF(1)=-1 THEN STOP
380     GOTO 340
390' ---------------------------------------------------------------------
400'
410'RUN
420'
430'    NOME ROSSI
440'    TELEFONO 02-631528
450'    NOME BIANCHI
460'    TELEFONO 02-646412
470'    NOME
480'
490'    PRIMA PASSATA
500'
510'    ROSSI        02-631528
520'    BIANCHI         02-646412
530'
560'    SECONDA PASSATA
570'
580'    ROSSI        02-631528
590'    BIANCHI         02-646412
```

Volendo scrivere il contenuto di più variabili in un unico record di un file sequenziale si deve usare l'istruzione PRINT# separando le variabili con un punto e virgola, come illustrato nell'esempio.

```
10          COGNOME$="ROSSI":NOME$="MARIO"
20          PRINT #1,NOME$;COGNOME$
 :            :
 :            :
100         INPUT #1,X$
110         PRINT X$

RUN

            MARIOROSSI
```

Separatori di records

Oltre ai caratteri speciali CR e LF per le stringhe di caratteri può essere utilizzata come separatore la virgola.

Fatta questa precisazione esaminiamo ora come avviene il processo di lettura di un record.

— Gli 'spazi bianchi', la 'virgola', il 'LF' ed il 'CR' all'inizio del record vengono ignorati.

— Se il primo carattere significativo è virgolette ("), vengono considerati come appartenenti al record tutti i caratteri che seguono fino al successivo ' " '.

— Se invece il primo carattere significativo non è ", la lettura termina quando viene incontrato un carattere separatore (, CR LF).

Attenzione: Se in una stringa di caratteri è contenuta una virgola e la stringa non è racchiusa tra virgolette, la lettura cessa appena viene incontrata la virgola, che è quindi considerata come elemento separatore.

Esempio:

```
10      X$="ROSSI,BIANCHI"
20      PRINT #1,X$
 :        :
 :        :
100     INPUT #1,X$
110     PRINT X$        ----------------->   ROSSI
```

LINE INPUT# n° file, variabile

Utilizzando questa opzione il sistema considera come elemento separatore solo il Carriage Return (CR). Qualora tale carattere non venga incontrato, la lettura viene arrestata automaticamente dopo 255 caratteri. Facendo riferimento all'esempio precedente, utilizzando

```
100 LINE INPUT#1,X$
```

si sarebbe ottenuto

```
110 PRINT X$ -----→ ROSSI,BIANCHI
```

Un altro modo di procedere consiste nell'assicurarsi che una stringa venga comunque racchiusa tra virgolette. Ricordando che il codice ASCII del carattere " è 34 si può pertanto ricorrere, per la scrittura in un file, alla forma:

```
PRINT # 1,CHR$(34);X$;CHR$(34)
```

Con questa istruzione infatti il record verrebbe formato nel modo seguente

```
[ "ROSSI, BIANCHI" ]
```

Attenzione!: PRINT#1,X$,Y$,Z$ scrive il contenuto delle variabili X$, Y$, Z$ separato da spazi bianchi (come LPRINT) e non da virgole.

Esempio:

```
100 X$="ROSSI":Y$="MARIO":Z$="631528"
110 PRINT#1,X$,Y$,Z$
```

dà origine al record seguente

```
| ROSSI   MARIO   631528 | CR | LF
```

In questo caso l'istruzione

```
INPUT# 1,X$
```

dà

```
X$="ROSSI   MARIO   631528"
```

Sottocampi

In un file sequenziale si possono immagazzinare dati anche nel seguente modo:

mediante la routine

```
100        INPUT NOM$,TEL$
 :                :
130        PRINT #1,NOM$;",";TEL$
 :                :
150        GOTO 100
```

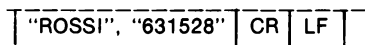
e procedere in seguito alla lettura in uno dei due modi qui esposti

```
      200 INPUT#1,NOM$,TEL$      200 LINE INPUT#1,X$
      210 PRINT NOM$,TEL$        210 PRINT X$
run
      ROSSI    631528            ROSSI,631528
```

WRITE# n° file,espressione1,espressione2,...(Microsoft 5.)

Utilizzando il WRITE al posto del PRINT si ottiene la separazione con virgole dei valori delle espressioni. Inoltre le stringhe di caratteri vengono automaticamente racchiuse tra virgolette.

Esempio:

```
100 NOME$="ROSSI": TEL$= "631528"
100 WRITE#1,NOME$,TEL$
```

DATI NUMERICI

Quando si memorizzano in un file sequenziale dei dati numerici, vengono adoperati come elementi separatori intermedi degli spazi bianchi. Utilizzando nell'istruzione PRINT, come separatore delle variabili, il ';' si ottiene la separazione dei valori, nel file, con un solo spazio bianco. L'uso del punto e virgola pertanto non permette per i dati numerici la concatenazione (come avveniva invece per le stringhe).

```
100 X=123:Y=456:Z=789
110 PRINT#1,X;Y;Z
```

I valori di X,Y,Z sono memorizzati in un record del file separati tra loro da uno spazio bianco. Tale spazio, in lettura, verrà poi considerato come separatore di records.

Pertanto

```
150 INPUT #1,X,Y,Z                lettura di 3 records
160 PRINT X,Y,Z
```

darà: 123 456 789

SCRITTURA DI STRINGHE E DATI NUMERICI

Esempio:

```
10      INPUT "RIFERIMENTO";REF$
20      INPUT "QUANTITA'";Q
30      PRINT #1',REF$
40      PRINT #1,Q
 :        :
200     INPUT #1,REF$
210     INPUT #1,Q
```

Attenzione! non fare:
```
10 X$="CICLO":X=123
20 PRINT#1,X$;X
```

poichè in questo caso la lettura di X$ darebbe come risultato 'CICLO 123'.

Simulazione di un file sequenziale con il terminale

```
10 INPUT"RECORD?";X$    'Lettura di uno pseudo-record'
20 PRINTX$
30 GOTO10
```

run

```
RECORD?XXX
XXX     (risultato)
----------------------
RECORD?YYY
YYY     (risultato)
----------------------
RECORD?ZZZ
ZZZ     (risultato)
```

CLOSE# n° file,#n° file, ...

In generale la scrittura su disco con i files sequenziali non viene fatta dopo ogni PRINT ma solo quando il buffer è pieno. Risulta pertanto molto importante chiudere i files al momento opportuno se non si vogliono perdere dei dati.

EOF (n° file)

Dà −1 se viene incontrata la fine del file durante un'operazione di lettura e 0 in tutti gli altri casi.

Esempio: IF EOF(1)=−1 THEN CLOSE#1:GOTO segue

LOC(n° file)

Fornisce il numero di blocchi di 128 bytes utilizzati per la lettura o la scrittura (INPUT o OUTPUT) (versione 5.).

INPUT$(X,n° file) Microsoft 5.

Permette la lettura di X caratteri da un file sequenziale. Utilizzando questa opzione vengono letti come caratteri normali anche i separatori di records.

Esempio: Dump di un file

```
10        OPEN "I",#1,"XX"
20        IF EOF(1)=-1 THEN CLOSE #1:STOP
30        PRINT HEX$(ASC(INPUT$(1,#1)));"      ";
40        GOTO 20
```

PRINT# n° file USING"format"; lista variabili

Le regole d'uso del PRINT#USING per i files sequenziali sono le stesse presentate per il classico comando PRINT USING.

AGGIUNTA DI RECORDS

Anche se taluni sistemi permettono di aggiungere dei records alla fine di un file sequenziale mediante la semplice riapertura del file stesso, in genere questo tipo di operazione non è consentito.

Nella maggior parte dei casi infatti per poter effettuare l'aggiornamento di un file occorre generare un nuovo file di tipo sequenziale in cui vengono scritti tutti i records di quello vecchio (che vengono man mano letti mediante INPUT#) seguiti poi dai records che si vogliono aggiungere. Questo metodo di aggiornamento permetterebbe, tra l'altro, anche l'eventuale modifica di alcuni records del vecchio file.

Questo sistema comunque non semplifica certamente l'uso dei files sequenziali, già di per sè molto laboriosi.

Nota: il sistema DTC Microfile ammette per i files di tipo sequenziale non solo l'aggiunta di records in coda ma anche il loro inserimento o la loro cancellazione. (Queste ultime operazioni vengono effettuate senza reali spostamenti fisici di records in quanto il sistema li concatena tramite dei puntatori).

CAPITOLO 4

QUALCHE SEMPLICE METODO PRATICO

DATA DEL FILE

Lavorando con dei files è importante conoscere la data in cui è stata effettuata l'ultima variazione. Questa informazione può essere memorizzata nel primo record del file ed aggiornata ogni volta che vengono fatte delle modifiche nel file stesso.

		G$	M$	A$	
DATA	1	9	1	81	
	2	ROSSI		XXXX	
	3	BIANCHI		XXXX	

```
10      OPEN "R",#1,"CLIENTI"
20      FIELD #1,1 AS G$,1 AS M$,1 AS AN$,36 AS BLK$
30      FIELD #1,15 AS CGN$,30 AS INF$
40      GET #1,1                                    'LETTURA PRIMO RECORD
50      PRINT USING "###";ASC(G$),ASC(M$),ASC(AN$)
                                                    'VISUALIZZAZIONE DATA
60      INPUT"OCCORRE AGGIORNARE LA DATA? ";R$
70      IF R$<>"SI" THEN GOTO 100
80      INPUT "NUOVA DATA";X,Y,Z
90      LSET G$=CHR$(X):LSET M$=CHR$(Y):LSET AN$=CHR$(Z)
100     PUT #1,1                                    'MEMORIZZAZIONE DELLA
105                                                 'NUOVA DATA'
110     DATA$=STR$(ASC(G$))+STR$(ASC(M$))+STR$(ASC(AN$))
120     PRINT DATA$                                 'VISUALIZZAZIONE NUOVA
130'                                                 DATA'
```

Si possono prevedere due date distinte: una relativa alla generazione del file e l'altra relativa all'ultimo aggiornamento.

Vale la pena di notare che, utilizzando il metodo di trasformazione proposto nell'esempio, sono sufficienti 3 bytes per la memorizzazione di una data.

STAMPA ORDINATA DI UN FILE AD ACCESSO DIRETTO

Poichè in genere non è possibile trasferire tutto il file in memoria, date le dimensioni limitate della memoria centrale, per effettuare un ordinamento sugli elementi del file bisogna procedere nel seguente modo:

1) Si effettua una lettura sequenziale del file conservando in memoria, per ogni record, solo il contenuto del campo (chiave) rispetto al quale si desidera ordinare il file. Le chiavi vengono via via memorizzate in un vettore, detto appunto vettore delle chiavi, mentre in un vettore parallelo (degli indici) verranno memorizzati i numeri dei records corrispondenti.

2) Si effettua poi un ordinamento sul vettore delle chiavi con uno dei metodi classici (ad esempio per inversioni successive) avendo cura di effettuare le stesse operazioni di inversione anche sul vettore degli indici.

3) Utilizzando alla fine il vettore degli indici in modo sequenziale per la ricerca dei records nei file si otterrà la lettura dei records nell'ordine desiderato.

Memorizzando, dopo l'ordinamento, il vettore degli indici così ottenuto sul disco ed avendo cura di effettuare sullo stesso gli opportuni aggiornamenti relativi all'aggiunta o all'eliminazione di records nel file, si può ottenere la stampa ordinata in 'tempo reale'.

Non ci soffermiamo sulla trattazione teorica del metodo 'ripple', riteniamo però opportuno fare alcuni accenni ad altri due metodi utilizzati spesso per ottenere l'ordinamento di tabelle: il metodo 'bubble' ed il metodo di Shell - Metzner.

Metodo 'bubble'

Questo metodo consiste nel confrontare via via ciascun elemento della tabella con tutti quelli che lo seguono effettuando tutte le inversioni che si rendono necessarie. In questo modo dopo la prima "passata" l'elemento minore sarà sicuramente al primo posto; dopo la seconda vi sarà al secondo posto l'elemento più piccolo tra i rimanenti e così via.

Dopo N−1 esplorazioni della tabella (se N è il numero degli elementi) si sarà ottenuto l'ordinamento voluto.

```
10     FOR I=1 TO N-1                                  'BUBBLE SORT'
20            FOR J=I+1 TO N                           '============'
30                  IF A(J)>A(I) THEN SWAP A(J),A(I)   'CONFRONTO TRA
31                                       'I-ESIMO ELEMENTO E TUTTI I
33                                                      'SEGUENTI'
40            NEXT J
50     NEXT I
```

Utilizzando questo metodo, il numero di confronti e di inversioni necessario per ottenere l'ordinamento di una tabella contenente valori casuali è dello stesso ordine di grandezza di quello relativo al metodo 'ripple'; tuttavia il tempo di esecuzione è senza dubbio inferiore in quanto non si rende necessario il controllo sul numero di inversioni effettuate ad ogni passaggio.

Metodo di Shell - Metzner

Questo metodo rappresenta una variante a quello di Shell da noi presentato come esempio alla fine della prima parte di quest'opera e permette di ridurre di un terzo il numero di confronti necessari per ottenere l'ordinamento.

```
100     N=100:PAS=N                          'SHELL-METZNER SORT'
105'                                         -------------------
110     PAS=INT(PAS/2):IF PAS<1 THEN STOP
120     J=1:K=N-PAS
125'
130     I=J
135'
150     L=I+PAS
160     IF A(I)<A(L) THEN 200
170     SWAP A(I),A(L)
180     I=I-PAS:IF I<1 THEN GOTO 200 ELSE GOTO 150
190'
200     J=J+1:IF J<K THEN 110 ELSE GOTO 130
```

Se la chiave è di tipo alfanumerico potrebbe sembrare utile fare ricorso ad un indice (puntatore alla tabella delle chiavi) in modo da effettuare le inversioni sui puntatori anzichè sulle chiavi. In realtà l'unico vantaggio sarebbe quello di non alterare la tabella delle chiavi, in quanto in tempo necessario per l'ordinamento rimarrebbe invece invariato.

Di seguito riportiamo in una tabella i tempi che si sono resi necessari, su un sistema 8080, per ottenere l'ordinamento di un vettore di numeri casuali compresi tra 0 e 1 (i tempi sono esposti in secondi).

	BUBBLE	SHELL	METZNER	INSERIMENTO DICOTOMICO
50 :	15	15	9	15
100 :	55	45	21	45
200 :	215	110	50	145
300 :	470	165	80	300
400 :			125	470
500 :			145	760

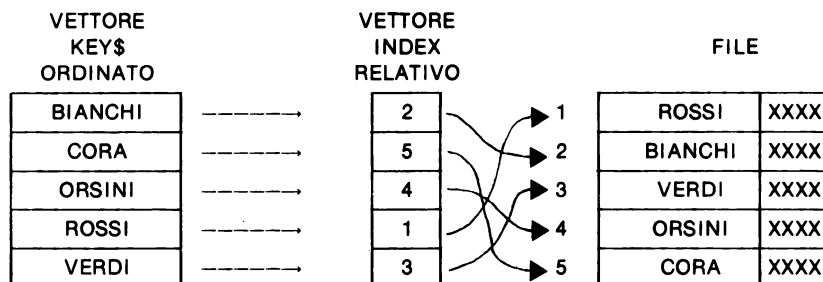
Per una tabella di 500 nomi di 12 caratteri il tempo necessario per l'ordinamento (con Shell - Metzner) sale da 145 sec. a 165 sec.

Inserendo nella tabella l'ultima colonna abbiamo voluto accennare al problema relativo all'aggiunta, in una tabella ordinata, di un nuovo elemento. Notiamo a questo proposito che per risolvere questo problema (ricerca della posizione in cui inserire il nuovo elemento) risulta in genere più conveniente utilizzare il metodo 'ripple' applicato in due sensi, piuttosto che ricorrere al metodo Shell.

```
10'       STAMPA ORDINATA DI UN FILE AD ACCESSO DIRETTO
20'       ===============================================
30'
40'       NEL CORSO DI UNA PRIMA LETTURA SEQUENZIALE DEL FILE SI CON-
50'       SERVANO IN MEMORIA, IN DUE VETTORI, SOLO LE CHIAVI PER
60'       L"ORDINAMENTO (AD ESEMPIO I "COGNOMI") E I NUMERI DEI
70'       RISPETTIVI RECORDS (VETTORI "KEY$" E "INDEX"). DOPO AVER
80'       OPPORTUNAMENTE ORDINATO QUESTI DUE VETTORI SI EFFETTUA UNA
85'       NUOVA LETTURA DEL FILE, TRAMITE IL VETTORE "INDEX" CHE SI
86'       PERCORRE IN MODO SEQUENZIALE, CON LA STAMPA DEL CONTENUTO
90'       DEI SINGOLI RECORDS. SI OTTIENE COSI" UN ELENCO DEI RECORDS
100'      ORDINATI SECONDO I VALORI DELLE CHIAVI.
105'
110'      N$: COGNOME      PN$: NOME        TL$: TELEFONO
115'
120       OPEN "R",#1,"CLIENTI"              'APERTURA DEL FILE'
130       FIELD #1,10 AS N$,8 AS PN$,20 AS TL$   'DEF CAMPI'
140' ------------------------------------------------------------------
150'          COSTRUZIONE DEI VETTORI    KEY$     INDEX
155'
160       NREC=LOF(1)-1:NKEY=0               'NREC: NUMERO RECORDS'
170       DIM KEY$(NREC),INDEX(NREC)
180'
190       FOR I=1 TO NREC
200                 GET #1,I                 'LETTURA DI UN RECORD'
210                 IF ASC(N$)=0 THEN 240          'RECORD VUOTO'
220                 NKEY=NKEY+1:KEY$(NKEY)=N$:INDEX(NKEY)=I
240       NEXT I
250' ------------------------------------------------------------------
260'      ORNDINAMENTO DEI VETTORI (METODO RIPPLE)
265'
270       K=NKEY
280       INVERSIONI=0
290'
300       FOR I=1 TO K-1
310                 IF KEY$(I+1)<KEY$(I) THEN
                     SWAP KEY$(I+1),KEY$(I):SWAP INDEX(I),INDEX(I+1):
                     INVERSIONI=1
320       NEXT I
330'
340       IF INVERSIONI=1 THEN K=K-1:GOTO 280
350'
360' ------------------------------------------------------------------
370'                                STAMPA
375'
380       LPRINT "ELENCO CLIENTI"
390       LPRINT "---------------":LPRINT"":LPRINT
400       FOR I=1 TO NKEY
410                 GET #1,INDEX(I)
430                 LPRINT N$,PN$,TL$
440       NEXT I
```

Supponendo ad esempio di avere nel file:

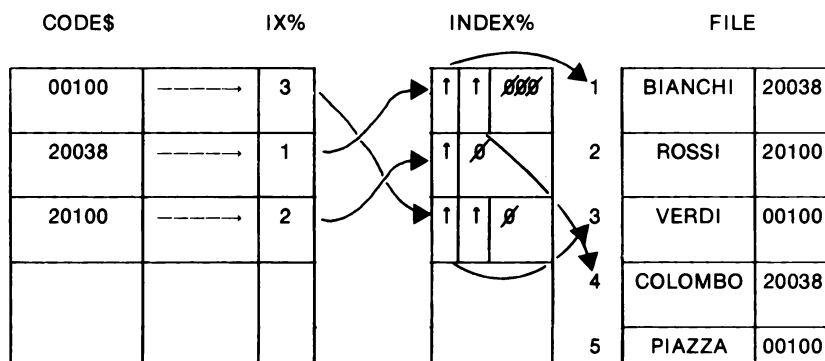
rec 1:	ROSSI ALBERTO	02 – 451412
rec 2:	BIANCHI GIORGIO	01 – 361824
rec 3:	VERDI ELVIRA	02 – 661522
rec 4:	ORSINI BRUNO	02 – 6427123
rec 5:	CORA SONIA	02 – 652313

dopo l'ordinamento dei vettori KEY$ e INDEX si avrebbe la configurazione illustrata in figura

che darebbe origine alla stampa ordinata.

```
10'     RAGGRUPPAMENTO PER CODICI
20'     ========================
30'
40'     QUESTO PROGRAMMA PERMETTE DI OTTENERE L"ELENCO DEI CLIENTI
50'     RAGGRUPPATI PER "CODICE POSTALE". PER OTTENERE QUESTO
60'     RAGGRUPPAMENTO SI E" FATTO RICORSO AD UNA MATRICE INDEX%
70'     COSTITUITA IN MEMORIA CENTRALE, CONTENENENTE IN CIASCUNA
80'     RIGA DEI PUNTATORI A TUTTI I RECORDS DEL FILE "CLIENTI"
90'     CONTENENTI LO STESSO VALORE NEL CAMPO CAP$.
100'    PER OTTENERE L"ELENCO SONO NECESSARIE DUE LETTURE DEL FILE
110'    UNA,SEQUENZIALE, PER LA CREAZIONE DELLA TABELLA INDEX%;
120'    LA SECONDA,RANDOM, PER LA STAMPA.
130'    PER OTTENERE UN ORDINAMENTO RISPETTO AI VALORI DEI CAP
140'    SI FA RIFERIMENTO AD ALTRE DUE TABELLE: CODE$, CONTENENENTE
150'    TUTTI I CODICI POSTALI AMMESSI, E IX%, CONTENENTE DEI
160'    PUNTATORI ALLE VARIE RIGHE DI INDEX% E CHE VIENE INIZIALIZZATA
170     CON I VALORI 1,2,3,....
180'
200'
210     DIM INDEX%(50,20),CODE$(50),IX%(50) 'MASSIMO 50 CODICI'
220     OPEN "R",#1,"CLIENT":FIELD #1,12 AS CL$,5 AS CAP$
230'-----------------------------------------------------------------
330     NC=0
340     FOR NR=1 TO LOF(1)-1                'COSTRUZIONE TABELLE'
350        GET #1,NR:IF ASC(CL$)=0 THEN 480
360'
370        FOR K=1 TO 50                    'TABELLA CODE$'
380               IF CAP$=CODE$(K) THEN 430
```

```
390                    IF CODE$(K)="" THEN CODE$(K)=CAP$:IX%(K)=K:
                              NC=NC+1:GOTO 430
400         NEXT K
410'
430         FOR I=1 TO 20                          'TABELLA INDEX%'
440                IF INDEX%(K,I)=0 THEN INDEX%(K,I)=NR:GOTO 480
450         NEXT I
460'
480      NEXT NR
490' ----------------------------------------------------------------
500      FOR I=1 TO NB-1                              'ORDINAMENTO CODE$'
510         FOR J=I+1 TO NR          'BUBBLE SORT'
520                IF CODE$(I)>CODE$(J) THEN
                      SWAP CODE$(I),CODE$(J):SWAP IX%(I),IX%(J)
530         NEXT J
540      NEXT I
550' ----------------------------------------------------------------
560      LPRINT:LPRINT"CAP";TAB(20) "CLIENTE":LPRINT:LPRINT
                                  'STAMPA'
570      FOR CODE=1 TO NC
580         LPRINT CODE$(CODE)
590'
600         FOR I=1 TO 20
610               N=INDEX%(IX%(CODE)):IF N=0 THEN 650
620               GET #1,N:LPRINT TAB(20) CL$
630         NEXT I
640'
650         LPRINT
660      NEXT CODE
```

Esempio esplicativo:

La figura illustra la situazione dopo l'ordinamento di CODE$

La stampa ordinata avrebbe dato origine al seguente elenco:

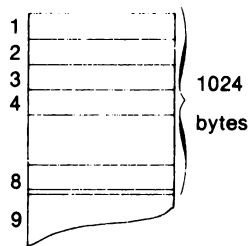
00100	VERDI
	PIAZZA
20038	BIANCHI
	COLOMBO
20100	ROSSI

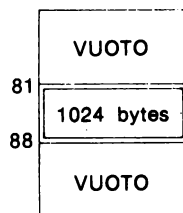
ALLOCAZIONE DELLO SPAZIO SU DISCO PER I FILES RANDOM

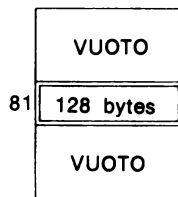
Per permettere una gestione più razionale della memoria di massa da parte del sistema, l'assegnazione ad un file di una porzione di disco avviene per blocchi di 1024 bytes (i blocchi assegnati possono eventualmente non essere contigui).

Ad ogni file viene associata una tabella contenente dei puntatori ai vari blocchi ad esso assegnati in modo che il sistema possa facilmente individuare le parti di disco occupate dal file.

Così se, ad esempio, viene creato il record 1 di 128 bytes relativo ad un nuovo file, ad esso vengono assegnati 1024 bytes (che potranno essere utilizzati per i records da 1 a 8). Quando in seguito si vorrà creare un nuovo record verranno assegnati al file altri 1024 bytes.

Con la versione 5. la generazione del record 81 in un file vuoto provoca l'allocazione solo di 1K di memoria. Se i records hanno lunghezza 128 bytes lo spazio riservato in memoria sarà utilizzabile per i records da 81 a 88.

Allocazione dello spazio su disco

Sul TRS80 NEW DOS la stessa operazione provoca invece l'allocazione di tutto lo spazio per generare anche gli 80 records precedenti.

Allocazione con microsoft 5.

Il sistema DTC Microfile invece assegna lo spazio su disco per blocchi di 128 bytes e gestisce tale spazio mediante dei puntatori interni. Ciò permette di avere sempre una gestione ottimizzata dello spazio allocato (cfr files virtuali).

Allocazione con DTC Microfile

ALLOCAZIONE DINAMICA

Utilizzando dei files ad accesso diretto è possibile effettuare l'aggiunta di nuovi records in coda al file stesso (utilizzando ad esempio la funzione LOF), mentre non è possibile recuperare dinamicamente lo spazio lasciato libero da eventuali records soppressi senza riorganizzare completamente l'intero file.

Risulta evidente a questo punto che una tale operazione, effettuata ogni qualvolta viene soppresso un record, produce una notevole perdita di tempo. Non converrebbe invece organizzare il file in modo da recuperare "dinamicamente" lo spazio disponibile senza riorganizzare "fisicamente" la struttura del file stesso?

Per ottenere questo tipo di organizzazione si possono seguire due vie: la concatenazione dei records resisi "disponibili" e la gestione di una Bit-Map. In questo paragrafo limiteremo la nostra trattazione al secondo metodo che ci sembra più significativo.

La Bit-Map è una tabella che specifica per ogni record del file se questo è utilizzato oppure se è disponibile.

Analizziamo ora nei dettagli la struttura di questa Bit-Map. Supponiamo che tale area di memoria sia formata da 256 bytes (cosa che permette di memorizzarla facilmente nei primi records del file) inizializzati a 0 in fase di inizializzazione.

Se il numero di records previsti per il file non supera 254 verrà associato ad ogni record un byte della Bit-Map. Ogni volta che viene utilizzato un record (generazione). viene posto a '1' il byte corrispondente nella Bit-Map, come illustrato nell'esempio:

	1	2	3	4	5	...	
1		1	1	Ø	1	1	
2	BIANCHI			XXXX			
3	ROSSI			XXXX			
4	libero						
5	COLOMBO			XXXX			

BIT – MAP

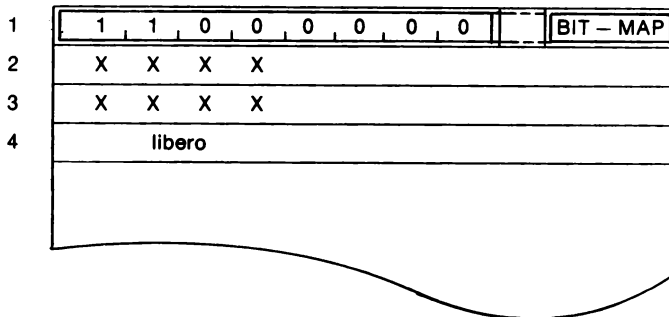
```
10       NREC=254:DIM BITMAP$(NREC)
20       FOR I=1 TO NREC:FIELD #1,(I-1)*1 AS D$,1 AS BITMAP$(I),
                   (256-I) AS R$:NEXT I
30       FIELD #1,12 AS N$,244 AS X$
40'
100      GOSUB 1000                    'RICERCA DI UN RECORD LIBERO'
110      INPUT "NUOVI DATI";NOME$,XX$
120      LSET N$=NOME$:LSET X$=XX$
130      PUT #1,RECLIB                 'RECLIB: INDICE RECORD LIBERO'
140      GOSUB 2000                    'AGGIORNAMENTO BITMAP'
150      STOP
990' ----------------------------------------------------------------
1000     GET #1,1                      'LETTURA DELLA BITMAP'
1005'
1010     FOR W=2 TO NREC               'RICERCA RECORD LIBERO'
1020       IF ASC(BITMAP$(W))=48 THEN RECLIB=W:GOTO 1500
1030     NEXT W
1040     PRINT "NON VI SONO RECORDS DISPONIBILI":STOP
1500     RETURN
1990' ----------------------------------------------------------------
1999                                   'AGGIORNAMENTO BITMAP:AGGIUNTA'
2000     GET #1,1:LSET BITMAP$(RECLIB)=CHR$(49):PUT #1,1:RETURN
2010' ----------------------------------------------------------------
2999                                   'SOPPRESSIONE
3000     GET #1,1:LSET BITMAP$(RECLIB)=CHR$(48):PUT #1,1:RETURN
3010' ----------------------------------------------------------------
```

Se invece si deve utilizzare un numero piuttosto alto di records conviene assegnare ad ogni record un solo bit nella Bit-Map.

In questo modo con 255 bytes si possono gestire più di 2000 records.

```
10'      GESTIONE DI UNA BITMAP
20'      ======================
30'
40       DIM BMAP$(50)
50       OPEN "R",#1,"BIT":FIELD #1,12 AS N$,244 AS X$
60       FOR I=1 TO 50:FIELD #1,(I-1) AS D$,1 AS BMAP$(I):NEXT I
70'
80       INPUT "NOME";NOME$
90       GOSUB 140
100      LSET N$=NOME$:PUT #1,RECLIB
110'
120' ----------------------------------------------------------------
130                                    'RICERCA RECORD LIBERO'
140      GET #1,1                      '---------------------'
150      FOR I=1 TO 50                 NBY:N.BYTE; NBIT:N.BIT
160        X%=ASC(BMAP$(I))
170        IF X%=255 THEN 250          'TUTTI I BIT SONO SETTATI'
180'                                   RICERCA DI UN BIT RESETTATO
190        FOR NBIT=1 TO 8
200            A%=2^(NBIT-1)
210            IF (X% AND A%)=0 THEN NBY=I:I=50:NEXT I:GOTO 290
220        NEXT NBIT
230'
250      NEXT I
```

```
260'
270     PRINT"NON VI SONO RECORDS DISPONIBILI":STOP
280'
290     LSET BMAP$(NBY)=CHR$(X%+A%)      'AGGIORNAMENTO DELLA BITMAP'
300     RECLIB=((NBY-1)*8+NBIT)+1         'RECORD LIBERO'
310     PRINT RECLIB
320     PUT #1,1
330     RETURN
340' ---------------------------------------------------------------
350                                       'SOPPRESSIONE DI UN RECORD'
360     GET #1,1
370     NBIT=(RECSOP-1) MOD 8:NBY=INT((RECSOP-1)/8)+1
380     LSET BMAP$(NBY)=CHR$(ASC(BMAP$(NBY))-(2^(NBIT-1)))
390     PUT #1,1
400     RETURN
410' ---------------------------------------------------------------
```

1	1 , 1 , 0 , 0 , 0 , 0 , 0 , 0	BIT – MAP
2	X X X X	
3	X X X X	
4	libero	

FILES RANDOM VIRTUALI (DTC MICROFILE)

L'utente vede questo tipo di file come un file ad accesso diretto, in realtà su disco viene allocato lo spazio necessario alla scrittura di un record solo all'atto del PUT. In questo modo si può generare logicamente un file di 4096 records vuoti senza occupare "fisicamente" spazio su disco. Infatti con questo tipo di gestione dello spazio se l'utente utilizza i records 2, 4 e 100 vengono occupati solo 3 'settori' sul disco.

Questo tipo di file, ad accesso Random 'virtuale' permette di ottimizzare lo spazio su disco e la ricerca del punto in cui allocare un record viene in genere fatta ricorrendo ai codici di tipo hash.

Organizzazione fisica

Una tabella indice permette di stabilire una corrispondenza tra i numeri dei records logici (codici conosciuti dall'utente) ed i reali indirizzi di tali records su disco.

In genere per ottimizzare il tempo di ricerca si fa ricorso a due livelli di indicizzazione: un indice a livello 0 la cui tabella è residente in memoria ed un indice di livello 1 la cui tabella, puntata 4095 tramite i_0 è residente su disco.

Questa seconda tabella è gestita dinamicamente ed è organizzata in blocchi di 128 bytes capaci quindi di contenere 64 puntatori.

Osserviamo che con una organizzazione di questo tipo non si possono gestire records con codice numerico superiore a 4096.

N PISTA INDEX N SETT.

N LOGICO

	INDEX
0	0
1	0
②	4,7
3	0
④	6,2
5	0
6	
7	
	0
4095	0

PISTA 4, SETTORE 7

② X X X X X

④ X X X X X

SETTORI ⟶

1 2 3 4 5 6 7 8

PISTE 1 2 3 4 5 0

1

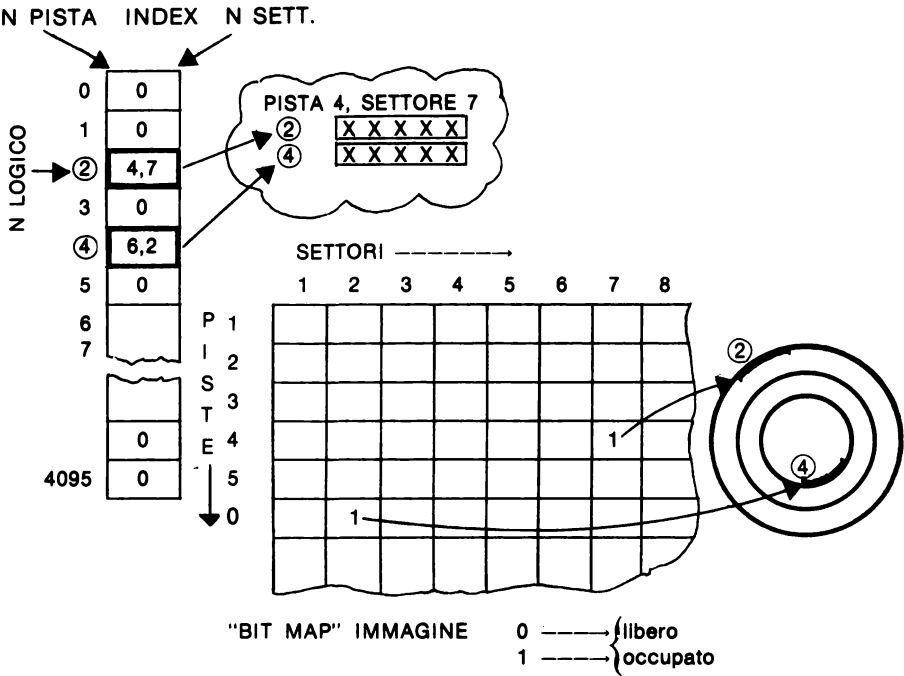
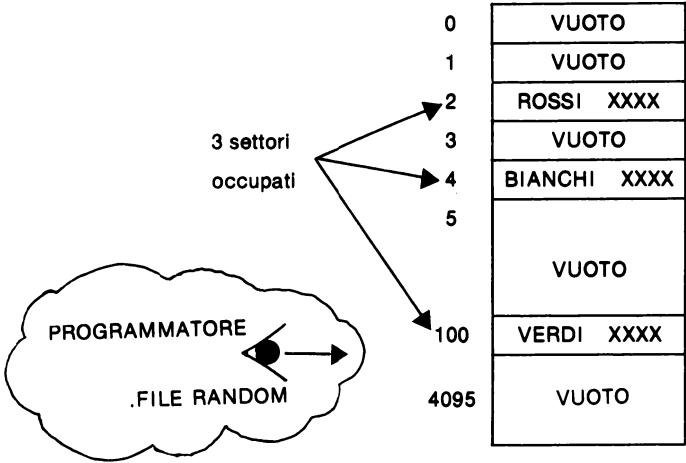
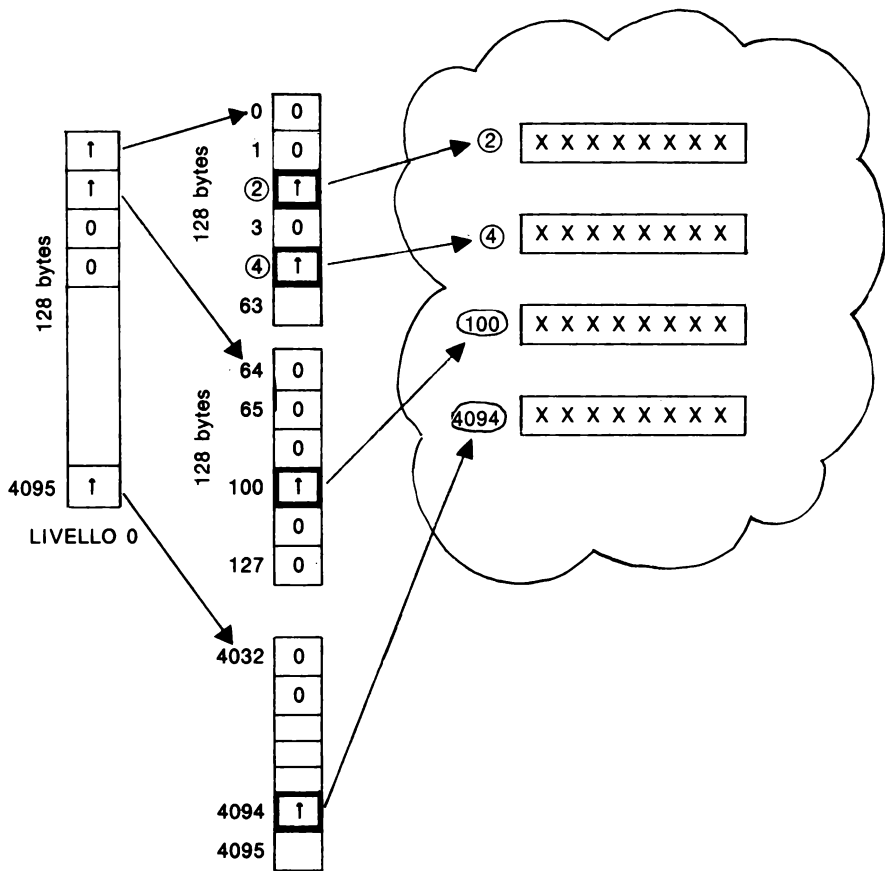
1

②

④

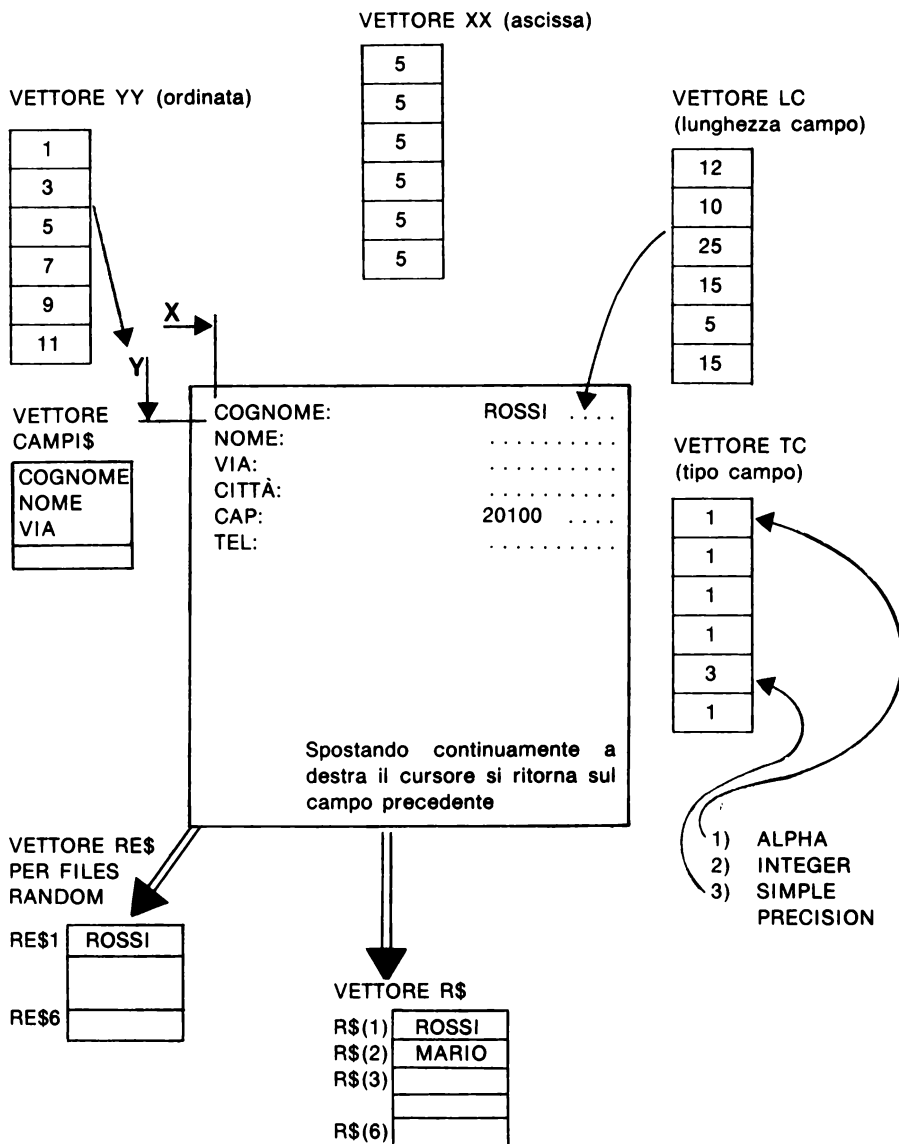
"BIT MAP" IMMAGINE

0 ⟶ libero

1 ⟶ occupato

ACCESSO INDICIZZATO (PRINCIPIO)

ACCESSO INDICIZZATO (REALE)

TERMINALE VIDEO TRS80

Durante una trasmissione di dati è importante poter controllare la correttezza di ciò che si trasmette e, possibilmente, effettuare questo controllo carattere per carattere.

Tale controllo viene ottenuto, sul TRS80, tramite la funzione INKEY$.

Il posizionamento del cursore su un punto del video può essere fatto mediante la funzione

```
PRINT @(Y*64)-X
```

ove X e Y rappresentano le coordinate del punto.

Il controllo sulla tastiera in attesa della battitura di un tasto viene fatto mediante la funzione INKEY$ come illustrato nell'esempio:

```
370 X$=INKEY$                  Se X$="" significa che non
380 IF X$=""THEN 370           è stato premuto nessun tasto
         :
         :
430 PRINT X$                   Viene visualizzato il carattere battuto
```

L'acquisizione di un'intera stringa viene ottenuta concatenando tra loro i caratteri battuti.

```
440 Y$=Y$+X$
```

L'eliminazione dell'ultimo carattere battuto si può ottenere battendo Control-H (codice ASCII 8) e viene realizzata nel modo seguente:

```
410 IF ASC(X$)=8 THEN Y$=LEFT$(Y$,LEN(Y$)-1)
```

Se per l'acquisizione dei caratteri viene utilizzata la funzione INKEY$ non si ha la gestione automatica del cursore. Il suo avanzamento o arretramento deve essere pertanto gestito da programma. Si può allora visualizzare il carattere acquisito nel punto in cui è posizionato il cursore mediante l'istruzione.

```
430 PRINT @XY + LEN (Y$),X$;
```

Se la gestione del cursore fosse invece automatica, per visualizzare il carattere X$ sarebbe sufficiente fare:

```
PRINT X$
```

(Per gestire la posizione del cursore su una riga si potrebbe ricorrere anche alle forme LSET e RSET).

L'esempio seguente, che fa riferimento alla figura della pagina precedente, mostra come viene realizzata la visualizzazione dei vari campi.

```
1540 ON TC(I) GOTO 1550,1560
1550      PRINT R$(I);STRING$(LC(I)-LEN(R$(I)),".")
1550      PRINT R$(I);STRING$(LC(I)-LEN(R$(I)),"-")
```

Se il valore da inserire in un campo supera la lunghezza massima prevista si ha un salto al campo successivo

```
450 IF LEN (Y$)>=LC(I) THEN GOTO 480
```

come risulta dal programma presentato in conclusione di questo paragrafo.

Mantenendo premuto il tasto RUBOUT si ottiene la cancellazione degli ultimi caratteri acquisiti, sempre che la stringa non sia vuota. Se invece si incontra una stringa vuota, premendo il tasto RUBOUT si ha un riposizionamento sul campo precedente.

Se si vuole utilizzare, per la memorizzazione dei dati acquisiti, un file ad accesso diretto, occorre dichiarare l'ampiezza dei campi (vettore RE$) tramite un'istruzione FIELD.

Nel nostro esempio sono previsti tre tipi di campi: 1 ALPHA – 2 INTEGER – 3 REAL SIMPLE P e la funzione di controllo è richiamata alla riga 420:

```
ON TC(I) GOSUB 580,590,590
```

Ai controlli da noi proposti possono essere aggiunti altri controlli "personalizzati", così come nulla impedisce di utilizzare dei particolari caratteri per cancellare un intero campo o per posizionarsi direttamente su un campo già acquisito.

Nel programma da noi proposto è prevista la visualizzazione del contenuto già esistente in un campo prima di accettarne la modifica.

Presentiamo ora il programma che realizza tutte le funzioni da noi descritte.

```
10'       ACQUISIZIONE DATI DA TERMINALE TRS80
20             CARATTERE PER CARATTERE
40'
70        CLEAR (3000):NC=6
80'
90'                                        RISULTATI IN R$ E RE$
100'
110'      INIZIALIZZAZIONE POSIZIONE DEL CURSORE
115'
120       DATA 1,3,5,7,9,11:FOR I=1 TO NC:READ YY(I):NEXT 'ORDINATE'
130       DATA 5,5,5,5,5,5: FOR I=1 TO NC:READ XX(I):NEXT 'ASCISSE'
140       DATA 1,1,1,1,3,1: FOR I=1 TO NC:READ TC(I):NEXT 'TIPO CAMPO
150       DATA 20,10,25,15,5,15:FOR I=1 TO NC:READ LC(I):NEXT
                                             'LUNGHEZZA DEI CAMPI'
160       DATA "COGNOME","NOME","VIA","CITTA","CAP","TELEFONO"
                                             'NOME DEI CAMPI'
170       FOR I=1 TO NC: READ CAMPI$(I):NEXT I
180'
190       GOSUB 250:STOP
210' ------------------------------------------------------------------
215'      GESTIONE DEL VIDEO                 PER I FILES RANDOM
220'                                         DICHIARARE I CAMPI
```

```
230'                                                    :
240'                                                    :
250       FOR I=1 TO NC:R$(I)="":NEXT
260       CLS                                  'CANCELLAZIONE VIDEO'
270       FOR I=1 TO NC                        'VISUALIZZAZIONE MASCHERA'
280          XY=YY(I)*64+XX(I):PRINT@ XY,CAMPI$(I);
295          PRINT@ XY+10,"";:ON TC(I) GOSUB 640,645,650
300          PRINT@ XY+34,"";:ON TC(I) GOSUB 550,560,560
310       NEXT I
320'
330       PRINT@ 924,"ROBOUT PER POSIZIONARSI SUL RECORD PRECEDENTE'
340       I=1:Y$=""
350       XY=YY(I)*64+XX(I)+34:IF TC(I)=1 THEN AE$="." ELSE AE$="-"
355'
360       PRINT@ XY+LEN(Y$),CHR$(136);    'POSIZIONAMENTO DEL CURSORE'
370       X$=INKEY$                            'CONTROLLO TASTIERA'
380       IF X$="" THEN 370
390       IF ASC(X$)=13 THEN IF Y$="" THEN GOTO 490 ELSE GOTO 480
                                               'CONTROLLO SUL CR'
400       IF ASC(X$)=8 THEN IF LEN(Y$)=0 THEN
             IF I>1 THEN GOSUB 540:I=I-1:GOTO 350 ELSE GOTO 370
410       IF ASC(X$)=8 THEN PRINT@ XY+LEN(Y$),AE$;:
             Y$=LEFT$(Y$,LEN(Y$)-1):GOTO 360
420       ON TC(I) GOSUB 580,590,590:ON Q GOTO 430,370 'CONTROLLO'
430       PRINT@ XY+LEN(Y$),X$;          'VISUALIZZAZIONE CARATTERE
440       Y$=Y$+X$                             'AGGIORNAMENTO STRINGA'
450       IF LEN(Y$)=>LC(I) THEN 480      'CONTROLLO SUL CAMPO'
460       GOTO 360
470'
480       R$(I)=Y$:ON TC(I) GOSUB 610,615,620:Y$="" 'MEMORIZZ.CAMPO
490       IF I>=NC THEN RETURN
510       GOSUB 540:I=I+1:GOTO 350          'RIPOSIZIONAMENTO'
530'
540       PRINT@ XY,"";:ON TC(I) GOSUB 550,560,560:RETURN
550       PRINT R$(I);STRING$(LC(I)-LEN(R$(I)),"."):RETURN 'VISUALIZZA-
560       PRINT R$(I);STRING$(LC(I)LEN(R$(I)),"-"):RETURN 'ZIONE
                                                             'STRINGA'
561'
562'
563'
580       IF ASC(X$)<32 THEN Q=2:RETURN ELSE Q=1:RETURN 'CONTROLLO
590       IF ASC(X$)<48 OR ASC(X$)>57 THEN Q=2:RETURN ELSE Q=1:RETURN
600'
610       LSET RE$(I)=Y$:RETURN                    'CODIFICA
615       LSET RE$(I)=MKI$(VAL(Y$)):RETURN        'DEI
620       LSET RE$(I)=MKS$(VAL(Y$)):RETURN        'CAMPI
635'
640       PRINT RE$(I);:RETURN                     'DECODIFICA
645       PRINT CVI(RE$(I));:RETURN                'DEI
650       PRINT CVS(RE$(I));:RETURN                'CAMPI
```

Gestione di un terminale remoto

Quando si deve utilizzare, per la trasmissione dei dati, un terminale remoto, cioè indipendente dal sistema, la gestione di quest'ultimo avviene tramite dei segnali di controllo che vengono trasmessi da e per l'U.C.E.

Per esempio per ottenere la *'cancellazione dello schermo'* si utilizza l'istruzione:

PRINT CHR$(X) dove il valore di X dipende dal particolare tipo di terminale.

Il *posizionamento del cursore* si ottiene inviando al terminale un carattere di controllo seguito dalle coordinate del punto:

Per alcuni terminali è necessario aggiungere alle reali coordinate una costante (32 p.es.)

Alcuni caratteri di controllo servono per gestire qualsiasi tipo di terminale:

per esempio:

PRINT CHR$(7)	attiva sempre la suoneria
PRINT CHR$(8)	fa retrocedere il cursore
PRINT CHR$(13)	riposiziona a capo

Nel sistema Microsoft 5. la funzione INPUT$(1) permette di acquisire i dati carattere per carattere.

Presentiamo ora in particolare la gestione di un terminale LX500.

I principali caratteri di controllo sono:

Cancellazione:	PRINT CHR$(133)
Inversione:	PRINT CHR$(29)
Fine video:	PRINT CHR$(28)
Posizionamento del cursore sul punto XY:	PRINT CHR$(27)+CHR$(61)+CHR$(32+Y) +CHR$(32+X)

Come si vede le coordinate vengono trasmesse *nell'ordine* Y,X.

Se si vuol generare un programma di acquisizione dati che valga per più tipi di terminali, dovrà essere predisposta una routine che assegni ai caratteri di controllo il valore adatto per ciascun terminale:

```
INPUT"TERM. TIPO";TYPE$
IF TYPE$="type1" THEN XY$=CHR$(XX1):...
IF TYPE$="type2" THEN XY$=CHR$(XX2):...
```

Presentiamo ora un programma per la gestione del terminale LX500 basato sui dati dell'esempio precedente.

Nota: per il codice postale (numerico) la lunghezza definita in FIELD(4) è diversa da quella definita per lo spazio su video (5 caratteri).

```
10'    GESTIONE DI UN TERMINALE REMOTO  LX500
35                                  'RISULTATI IN R$ E RESUL$
40     OPEN "R",#1,"FILE"
60     CANCEL$=CHR$(133):RETROCURS$=CHR$(8):XY$=CHR$(27)+CHR$(61):
       VIDEO$=CHR$(29):FVIDEO$=CHR$(28): BX=32:CR$=CHR$(13):
       DRIN$=CHR$(7)
80     NC=5:DIM RESUL$(NC),R$(NC),CAMPI$(NC)
90     DATA "COGNOME","NOME","VIA","CITTA","CAP":
       FOR I=1 TO NC:READ CAMPI$(I):NEXT I
100    DATA 2,2,2,2,2:FOR I=1 TO NC:READ XX(I):NEXT 'ASCISSE'
110    DATA 3,5,7,9,11:FOR I=1 TO NC:READ YY(I):NEXT 'ORDINATE'
120    DATA 1,1,1,1,3:FOR I=1 TO NC: READ TC(I):NEXT 'TIPO'
130    DATA 12,15,25,10,5:FOR I=1 TO NC:READ LC(I):NEXT 'LUNGHEZZA
150    FIELD #1,12 AS RESUL$(1),15 AS RESUL$(2),25 AS RESUL$(3),
                 10 AS RESUL$(4),5 AS RESUL$(5)
160    DEF FNXY$(X,Y)=XY$+CHR$(BX+Y);CHR$(BX+X) 'INDIRIZZ.CURSORE
170    GET #1,100:GOSUB 190:PUT #1,100:STOP
180'-------------------------------------------------------------------
190    PRINT CANCEL$;:FOR I=1 TO NC:R$(I)="":NEXT I
210    FOR I=1 TO NC
220        PRINT FNXY$(XX(I),YY(I));VIDEO$;:PRINT USING"       ";
                 CAMPI$(I);
230        PRINT FNXY$(XX(I)+15,YY(I));FVIDEO$;
240        ON TC(I) GOSUB 560,570,580       'VISUALIZZ. VECCHI VAL'
250        PRINT FNXY$(XX(I)+35,YY(I));
260        ON TC(I) GOSUB 450,460,460       'VISUALIZZ.MASCHERA'
270    NEXT I
290    I=1:Y$=""
300    PRINT FNXY$(XX(I)+35,YY(I));:IF TC(I)=1 THEN AE$="." ELSE
                                                         AE$="-"
320    X$=INPUT$(1)                   'LETTURA DI UN CARATTERE'
330    IF X$=RETROCURS$ AND LEN(Y$)=0 THEN PRINT DRIN$;:
          IF I>1 THEN GOSUB 440:I=I-1:Y$="":GOTO 300 ELSE GOTO 320
340    IF X$=RETROCURS$ THEN Y$=LEFT$(Y$,LEN(Y$)-1):
          PRINT RETROCURS$;AE$;:GOTO 320 'SOPPRESSIONE 1 CARATTERE
350    IF X$=CR$ THEN IF Y$="" THEN GOTO 410 ELSE GOTO 400
360    ON TC(I) GOSUB 480,490,490:ON Q GOTO 370,320
370    Y$=Y$+X$:PRINT X$;
380    IF LEN(Y$)<LC(I) THEN GOTO 320 ELSE PRINT DRIN$;
400    R$(I)=Y$:ON TC(I) GOSUB 510,520,530:Y$=""   'CODIFICA'
410    GOSUB 440:IF I>NC THEN RETURN
420    I=I+1:GOTO 300
440    PRINT FNXY$(XX(I)+35,YY(I));:ON TC(I) GOSUB 450,460,460:RETURN
450    PRINT R$(I);STRING$(LC(I)-LEN(R$(I)),"."):RETURN
460    PRINT R$(I);STRING$(LC(I)-LEN(R$(I)),"-"):RETURN
470'
480    IF ASC(X$)<32 THEN PRINT DRIN$;:Q=2:RETURN 'CONTROLLO'
                   ELSE Q=1:RETURN
490    IF ASC(X$)<48 OR ASC(X$)>57 THEN PRINT DRIN$;:Q=2:RETURN
                                      ELSE Q=1:RETURN
500'
510    LSET RESUL$(I)=Y$:RETURN
520    LSET RESUL$(I)=MKI$(VAL(Y$)):RETURN          'CODIFICA'
530    LSET RESUL$(I)=MKS$(VAL(Y$)):RETURN
560    PRINT RESUL$(I);:RETURN                      'DECODIFICA'
570    PRINT CVI(RESUL$(I));:RETURN
580    PRINT CVS(RESUL$(I));:RETURN
```

CAPITOLO 5

L'ACCESSO INDICIZZATO

I files ad accesso diretto, il cui uso è ammesso da quasi tutti i sistemi Personal in commercio, permettono di accedere ad un record solo tramite un indice posizionale. In genere però, specialmente in ambito gestionale, è utile accedere ai records dei files, anche tramite dei codici particolari, detti chiavi (codici mnemonici, nomi simbolici od altro).

La ricerca di un record in un file tramite una chiave (che in genere non è altro che il contenuto di un 'campo') può ovviamente essere fatta mediante una lettura sequenziale del file stesso. Tenendo però conto del tempo necessario per la lettura di un record (circa 100 ms) risulta troppo alto il tempo di risposta per files con un numero di records piuttosto alto.

Conviene quindi, anzichè seguire questa strada, stabilire una corrispondenza tra il valore della chiave e la posizione occupata dal record nel file, in modo da ridurre sensibilmente il numero di accessi al disco.

Per realizzare questo tipo di corrispondenza si utilizza, in genere, uno di questi due criteri:
- ricavare l'indirizzo fisico del record tramite una funzione che permette, in un numero ridotto di passaggi, di calcolarlo direttamente dal valore della chiave (codifica hash).
- utilizzare una tabella indice di conversione.

Il primo metodo, pur essendo di facile applicazione, è poco usato in quanto richiede l'allocazione di molto spazio su disco anche per files piuttosto ridotti. Per questo motivo centreremo la nostra attenzione sui files ad accesso 'indicizzato' ed in particolare prenderemo in considerazione questi tre metodi:

- accesso indicizzato semplice in cui la tabella di conversione viene ogni volta generata in memoria centrale

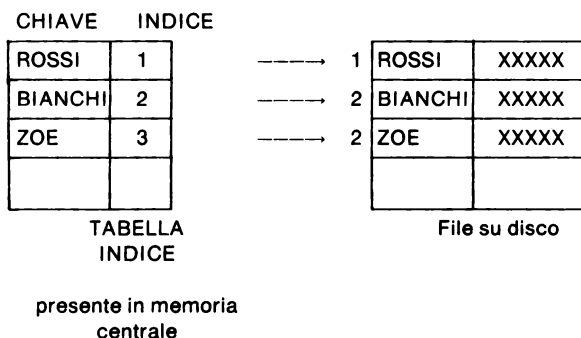
– accesso indicizzato con conservazione delle tabelle di conversione in un apposito file

– accesso indicizzato con l'uso di due livelli di indicizzazione.

La scelta tra questi tre metodi deve essere fatta dal programmatore tenendo conto delle particolari esigenze del problema da trattare.

Alla fine del capitolo viene proposto un esempio, realizzato sul TRS80, che illustra in modo abbastanza soddisfacente le tecniche per l'uso dei files indicizzati.

ACCESSO INDICIZZATO CON TABELLA DI CONVERSIONE IN MEMORIA CENTRALE

La tabella di conversione valore chiave-indice record è residente in memoria e viene generata, tramite una lettura sequenziale del file, all'inizio del programma.

Se la tabella di conversione non è ordinata rispetto ai valori della chiave, per accedere ad un record del file occorre effettuare prima una ricerca sequenziale della chiave nella tabella e poi accedere al file tramite l'indice corrispondente.

Poichè questo metodo richiede la costruzione della tabella di conversione (INDICE) all'inizio del programma, risulta essere poco conveniente quando il file contiene qualche centinaio di records: in questo caso infatti occorrerebbe un tempo piuttosto 'elevato' per la generazione della tabella stessa (qualche decina di secondi). Per contro l'indicizzazione semplice ha il vantaggio di essere facilmente gestibile anche da programmatori poco esperti e di adattarsi agevolmente anche a programmi già esistenti in quanto può essere realizzata tramite due semplici sottoprogrammi: uno per la generazione della tabella e l'altro per la ricerca sequenziale della chiave nella tabella stessa.

Per risparmiare spazio in memoria si potrebbe ottimizzare la costruzione della tabella memorizzando, ad esempio, solo i primi due caratteri relativi ad ogni valore della chiave. Questo artificio, che permette tra l'altro di ridurre notevolmente i tempi di ricerca, presenta però un inconveniente: se esistono più chiavi aventi i primi due caratteri in comune si rendono necessari più accessi al file per ritrovare il record desiderato.

Nel caso in cui un record logico sia suddiviso in più records fisici sarà necessario memorizzare nella tabella, per ogni valore della chiave, oltre al numero di record fisico corrispondente anche la posizione che il record logico desiderato occupa all'interno del record fisico. Queste due informazioni possono essere raggruppate in un unico dato così ottenuto:

N=(*n° rec. fisico**NLOG)+POS

ove NLOG rappresenta il numero di records logici contenuti in ogni record fisico e POS la posizione occupata dal record logico in esame all'interno del record fisico.

La decodifica successiva può essere fatta nel modo seguente:

NFIS=INT(N/NLOG)
POS=N MOD NLOG

Nell'esempio da noi proposto l'aggiornamento della tabella di conversione, con l'aggiunta di nuove chiavi, può venir fatto solo tramite una nuova lettura del file. Pertanto non sarà possibile accedere, tramite la tabella, ad un nuovo record per tutto il periodo intercorrente tra la generazione del record stesso e la rigenerazione della tabella di conversione.

```
10'      ACCESSO INDICIZZATO CON L"USO DI UNA TABELLA DI CONVERSIONE
20'      POSTA IN MEMORIA E GENERATA ALL"INIZIO DEL PROGRAMMA
30'
40'
70       OPEN "R",#1,"FILE"
90       FIELD #1,10 AS N$,15 AS CN$,20 AS TL$,4 AS COD$
100'
110      DIM NOM$(200),INDEX(200):GOSUB 190   'GENERAZIONE TABELLA'
120'
130      PRINT:INPUT"RICERCA PER NOME ";R$
140      IF R$="SI" THEN GOSUB 300
150      STOP
160'
170'     ------------------------------------------------------------------
180'                                      GENERAZIONE DELLA TABELLA INDICE
190      J=1                              '--------------------------------
200      FOR I=1 TO LOF(1)-1              'LETTURA SEQUENZIALE DEL FILE'
210          GET #1,I
220          IF ASC(N$)=0 THEN 240
230          NOM$(J)=CN$:INDEX(J)=I:J=J+1      'CORRISPONDENZA
```

```
240    NEXT I                                   'CHIAVE-INDICE'
250'
260    RETURN
270'---------------------------------------------------------------
290'                                  RICERCA PER CHIAVI
300    PRINT:INPUT"COGNOME ";KEY$:IF KEY$="" THEN RETURN
310'
320    L=LEN(KEY$)                    'SI POTREBBE ABBREVIARE IL COGNOME
330'
335'                                  'RICERCA CHIAVE'
340    FOR I=1 TO J-1                 'J-1=NUMERO DI CHIAVI'
350        IF KEY$=LEFT$(NOM$(I),L) THEN X=I:I=J-1:NEXT I:GOTO 410
360        IF I=J-1 THEN PRINT:PRINT"SCONOSCIUTO":PRINT:GOTO 300
370    NEXT I
380'
410    GET #1,INDEX(X)                'LETTURA RECORD CERCATO'
420    PRINT:PRINT N$,CN$,TL$
430    GOTO 300
440'------------------------------------------------------------
450'    *  USANDO IL SISTEMA MICROSOFT 5.
460'              FARE ATTENZIONE ALL"USO DI LOF
470'
480'    *  USANDO IL SISTEMA TRS80
490'              USARE: CLEAR(3000)
                  RIMPIAZZARE LOF(1) CON LOF(1)+1
510'              PRIMA DI UN"ISTRUZIONE INPUT X$ ASSICURASI CHE X$=""
520'              PER ALLOCARE IL DISCO 1 USARE: OPEN "R","FILE:1"
530'              RICORDARSI DI CHIUDERE I FILES !!!!!!!!!!!!!!!
540'---------------------------------------------------------------
```

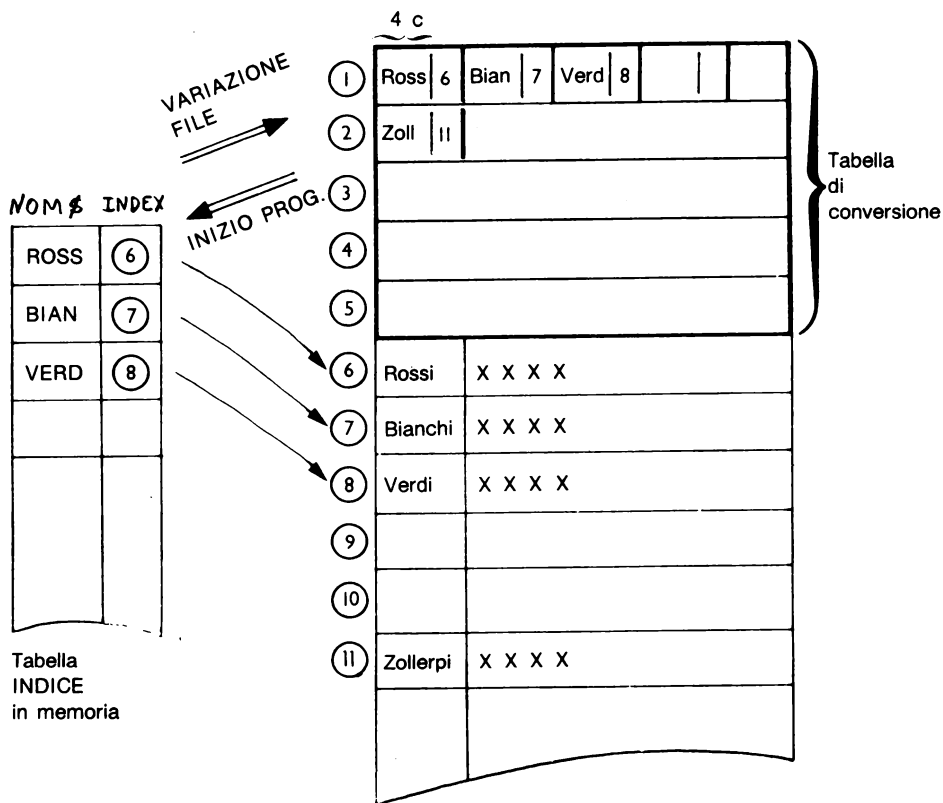
ACCESSO INDICIZZATO CON L'USO DI UNA TABELLA DI CONVERSIONE CONSERVATA NEL FILE

Se la tabella di conversione viene conservata nei primi records del file, effettuandovi eventualmente delle modifiche, basterà riportarla in memoria all'inizio di ogni seduta, risparmiando così il tempo necessario per la sua generazione. Bisogna soltanto fare attenzione ad effettuare l'aggiornamento della tabella residente nel file ogni qual volta si effettua una variazione (aggiunta o eliminazione di records) nel file stesso. Infatti senza questo accorgimento si rischierebbe di perdere tutte le informazioni relative alle modifiche effettuate in caso di interruzione accidentale nell'alimentazione.

Inoltre, se la tabella viene conservata in modo ordinato per permettere ad esempio la ricerca dicotomica, bisogna riorganizzare l'intera tabella ogni volta che si deve aggiungere una chiave. (Se il numero di records è piuttosto elevato questa operazione può risultare lunga).

Il programma seguente illustra i concetti sopra esposti.

```
10'   ACCESSO INDICIZZATO CON L"USO DI UNA TABELLA INDICE CONSERVATA
20'   SU DISCO
30'
40'
90    NKEY=20:LKEY=4                  'SUL TRS80: NKEY=40'
100   DIM KEY$(NKEY),PT$(NKEY)        'LKEY:LUNGHEZZA CHIAVE
110   OPEN "R",#1,"FILE"              'NKEY:NUMERO DI CHIAVI PER RECORD'
120'
130   FIELD #1,10 AS N$,15 AS CN$,20 AS TEL$,4 AS COD$
140   FOR I=1 TO NKEY
145       FIELD #1,6*(I-1) AS D$,LKEY AS KEY$(I),2 AS PT$(I)
149   NEXT I
150   DIM NOM$(200),INDEX(200)
160'
170   GET #1,1:IF ASC(KEY$(1))=0 THEN GOSUB 960:GOTO 190
172              'INIZIALIZZAZIONE DELLA TABELLA INDICE'
180   GOSUB 500
190   PRINT TAB(20) "FUNZIONI":PRINT
200   PRINT TAB(20) "CREAT:GENERAZIONE DI UN NUOVO RECORD"
210   PRINT TAB(20) "RNI:RICERCA PER CHIAVE NOMINATIVA"
220   PRINT
230   PRINT:INPUT"FUNZIONE(CREAT,RNI,FINE) ";MODE$
240   IF MODE$="CREAT" THEN GOSUB 310
250   IF MODE$="RNI" THEN GOSUB 900
260   IF MODE$="FINE" THEN CLOSE #1:STOP
270   GOTO 230
280'
290'-------------------------------------------------------------------
300'                              GENERAZIONE
310   GOSUB 740:ON Q GOTO 320,330,460      'CONTROLLO SULLA CHIAVE
320   PRINT "ESISTE GIA'":GOTO 310          'Q=1
330   NREC=LOF(1):GET #1,NREC               'AGGIUNTA ALLA FINE FILE
340   LSET CN$=NOM$
350'
360   INPUT"NOME ";NOME$:LSET N$=NOME$
370   INPUT"TELEFONO ";T$:LSET TEL$=T$
380   INPUT"MATRICOLA ";MT$:LSET COD$=MT$
390'
400   PRINT:PRINT CN$,N$,TEL$,COD$:R$=""
405   INPUT"OK? ";R$:IF LEFT$(R$,1)<>"S" THEN IF R$<>"OK" THEN 310
410   PUT #1,NREC
420   PRINT:PRINT"IL NUOVO CLIENTE E' MEMORIZZATO NEL RECORD ";NREC
430   NOM$(IND)=CN$:INDEX(IND)=NREC:IND=IND+1 'AGGIORNAMENTO TABELLE
440   GOSUB 1010                  'AGGIORNAMENTO FILE INDICE'
450   GOTO 310
460   RETURN
470'-------------------------------------------------------------------
480'                              'LETTURA DELLA TABELLA DA DISCO
490'
500   IND=1                      'PUNTATORE ALLA TABELLA DI CONVERSIONE
510   FOR I=1 TO 5               '5 RECORDS SONO ASSEGNATI ALLA TABELLA
520       GET #1,I
530'
540       FOR J=1 TO NKEY
550             IF ASC(KEY$(J))=0 THEN 570
560             NOM$(IND)=KEY$(J):INDEX(IND)=CVI(PT$(J)):IND=IND+1
570       NEXT J
580'
590   NEXT I
600   RETURN
```

```
610'
620'-----------------------------------------------------------------------
630'
640'
650'
660'                                  RICERCA DI CHIAVE:
670'                                   Q=1:CHIAVE TROVATA
680'                                   Q=2:CHIAVE INESISTENTE
690'                                   Q=3:FINE RICERCA
700'
740     PRINT:CGN$="":INPUT"COGNOME ";CGN$
750     IF CGN$="" THEN Q=3:RETURN
760     L=LEN(CGN$):W=1
770'
780     FOR I=W TO 200
790         IF NOM$(I)="" THEN GOTO 830
800         IF CGN$=LEFT$(NOM$(I),L) OR CGN$=LEFT$(NOM$(I),LKEY) THEN
                                  GOTO 850
810     NEXT I
820'
830     Q=2:RETURN
840'
850     GET #1,INDEX(I)              'CONTROLLO DELLA CHIAVE NEL RECORD
860     IF CGN$=LEFT$(CN$,L) THEN NREC=INDEX(I):Q=1:RETURN
870     W=I+1:GOTO 780               'NON CORRISPONDE NEL FILE'
880'
890'-----------------------------------------------------------------------
895'                                  'RICERCA INDICIZZATA
900     GOSUB 740:ON Q GOTO 910,920,930  'RICERCA DI CHIAVE
910     PRINT:PRINT N$,CN$,TEL$,COD$:GOTO 900  'VISUALIZZAZIONE RECORD
920     PRINT:PRINT"CLIENTE SCONOSCIUTO":GOTO 900
930     RETURN
940'-----------------------------------------------------------------------
950'                                       INIZIALIZZAZIONE
960'    INIZIALIZZAZIONE DELLA TABELLA CON VALORI NULLI
970     FOR I=1 TO 5:GET #1,1:PUT #1,I:NEXT I
980     RETURN
990'-----------------------------------------------------------------------
1000'                                 MEMORIZZAZIONE DELLA TABELLA SU DISCO
1010    K=1
1020    FOR I=1 TO 5
1030        GET #1,I
1040'
1050          FOR J=1 TO NKEY
1060              IF NOM$(K)="" THEN PUT #1,I:RETURN
1070              LSET KEY$(J)=NOM$(K):LSET PT$(J)=MKI$(INDEX(K)):
                                  K=K+1
1080          NEXT J
1090'
1100          PUT #1,I
1110    NEXT I
1120'-----------------------------------------------------------------------
1130'             * PER 5. VEDERE USO DELLA FUNZIONE LOF
1140'
1150'            * PER TRS80 NEWDOS
1160'                 CAMBIARE LOF(1) IN LOF(1)+1
1170'                 CLEAR(3000)
1180'                 PRIMA DI UN INPUT X$ PORRE X$=""
1190'                 OPEN "R","FILE,NUMERO DISCO"
1200'-----------------------------------------------------------------------
```

USO DI UN FILE INDICE ORDINATO

Quando il file ha dimensioni notevoli può essere antieconomico mantenere in memoria la intera tabella di conversione. In tal caso si consiglia invece di operare nel modo seguente:

- le chiavi vengono memorizzate, con i relativi puntatori, in un file ausiliario, detto file INDICE, e viene generata in memoria una tabella di conversione ridotta costruita nel modo seguente: si memorizza nella tabella solo la prima chiave contenuta in ciascun record del file INDICE. Poichè le chiavi nel file ausiliario sono disposte in ordine crescente, data una chiave qualsiasi è facile risalire al record di INDICE in cui essa è contenuta, tramite una ricerca nella tabella di conversione.

- viene poi fatta una ricerca della chiave richiesta all'interno del record di INDICE selezionato e si accede infine tramite il puntatore associato al file originario.

Con questo metodo quindi sono necessari due accessi al disco per individuare un record nel file originario.

L'uso di un file ausiliario "ordinato" può apparire piuttosto scomodo, dato che ogni volta che si richiede l'aggiunta di una nuova chiave occorre riorganizzarlo per mantenerne l'ordinamento.
Notiamo a questo proposito che il tempo necessario per l'inserimento di una nuova chiave nel file INDICE è inversamente proporzionale alla lunghezza dei records del file stesso. Infatti, a parità di numero di chiavi, tanto più i records sono lunghi, tanto meno è necessario traslarli.

Nonostante questo inconveniente il metodo presentato mantiene una certa validità in quanto, in genere, il numero di ricerche per chiavi è nettamente superiore a quello di eventuali aggiunte.

DOPPIO LIVELLO DI INDICIZZAZIONE

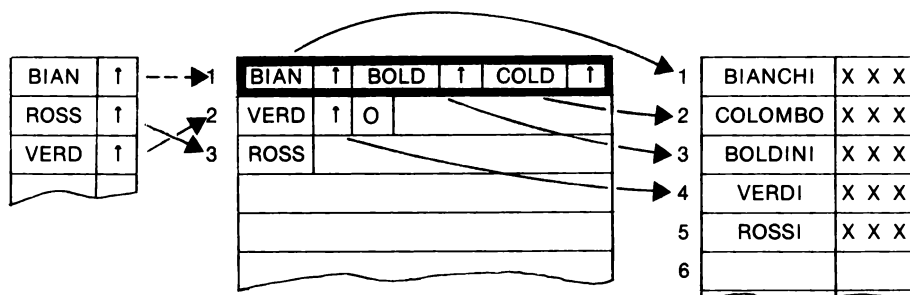
Per migliorare il metodo precedentemente illustrato si può ricorrere all'allocazione dinamica dei records del file INDICE. In questo caso si parla di doppio livello di indicizzazione.
Infatti in fase di generazione della tabella ridotta in memoria si associa ad ogni chiave il puntatore al record di INDICE in cui è contenuta (ricordiamo che nella tabella viene utilizzata solo la prima chiave di ogni record) in modo da mantenerla ordinata anche dopo l'aggiunta di nuovi elementi. Il file INDICE viene anch'esso costruito in modo dinamico man mano che si richiedono delle aggiunte.

In effetti quando si deve inserire nel file INDICE una nuova chiave si opera nel modo seguente:

- dopo aver individuato, tramite la tabella, il record in cui dovrebbe essere logicamente inserita la chiave si controlla se l'inserimento è possibile.
- se il record in esame è pieno, ma il successivo permette inserimenti, si trasla in questo record l'ultima chiave che era contenuta nel record pieno;
- se invece anche il record successivo è pieno si genera un nuovo record in cui viene inserita la chiave che logicamente doveva essere contenuta nel record in esame. L'ordinamento in ogni caso è assicurato dalla tabella di conversione (indice di livello 0) presente in memoria.

Si opera in pratica come se il file INDICE contenesse dei "buchi" per permettere gli inserimenti.

Con questo accorgimento si rende necessaria la manipolazione di due soli records del file INDICE per inserire una chiave.

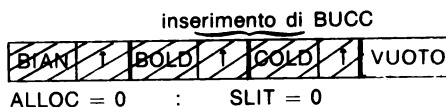
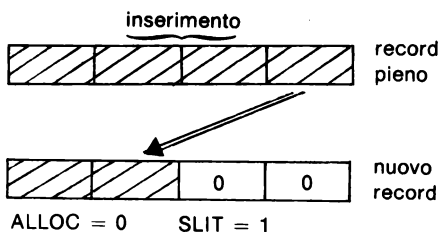
La figura seguente illustra l'uso di questo doppio livello di indicizzazione

- INDICE LIV. 0 TABELLA ORDINATA
- Contiene la prima chiave di ogni record

- INDICE DI LIV. 1 (FILE)
- Le chiavi sono ordinate all'interno di ogni record

- FILE PRINCIPALE

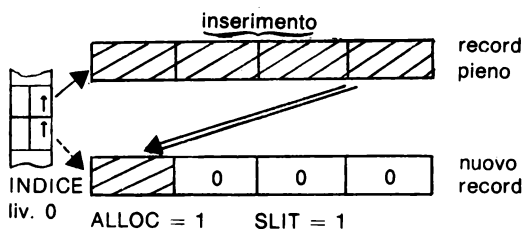
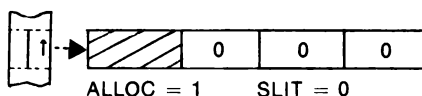
Qui di seguito le illustrazioni spiegano come avviene l'inserimento di una nuova chiave nel file INDICE, nei vari casi possibili.

ALLOC ———→ ALLOCAZIONE NUOVO RECORD
SLIT ———→ SLITTAMENTO TRA DUE RECORDS

— Vi è un posto libero nel record in cui deve essere inserita la chiave

— Vi è un posto libero nel record seguente

– Non vi è posto neppure nel record seguente:

→ Assegnazione dinamica di un nuovo record

\+ Slittamento di un dato tra due records

\+ Aggiornamento della tabella INDICE liv. 0.

– Aggiunta di un nuovo record

ACCESSO INDICIZZATO SUL TRS80

L'esempio da noi proposto illustra un metodo di indicizzazione piuttosto semplice da usare ma applicabile solo per un numero limitato di chiavi (300 circa).

Assegnando due records per ciascun gruppo di 2 lettere si potrebbe comunque raddoppiare la capacità del file INDICE.

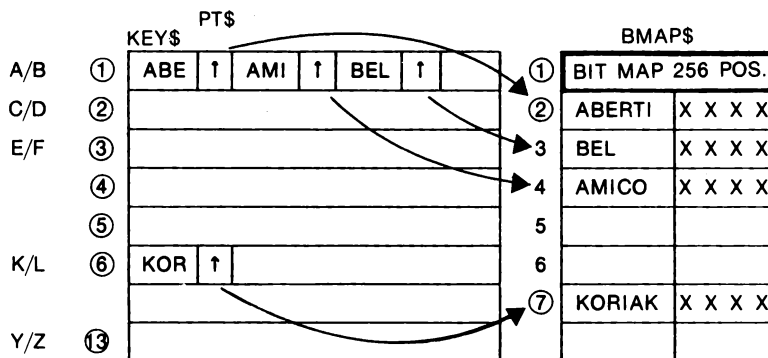
L'allocazione dinamica nel file è stata realizzata utilizzando una pseudo Bit-Map in modo da evitare la continua riorganizzazione del file stesso.

Una lettura sequenziale del file INDICE (in cui le chiavi sono ordinate) permette di ottenere un elenco ordinato dei nomi contenuti nel file principale.

```
10      FOR I=1 TO 13                         '13 RECORDS NEL FILE INDICE
20          GET #2,I
30          FOR J=1 TO MXNKEY                 'MXNKEY:N MASSIMO DI CHIAVI
                                               PER RECORD
40              IF CVI(PT$(J))=0 THEN GOTO 30
50              GET #1,CVI(PT$(J))
60              PRINT N$
70          NEXT J
80      NEXT I
```

Osserviamo che un indice distrutto può essere facilmente rigenerato mediante una lettura sequenziale del file principale, e in modo analogo può essere generata una Bit-Map.

```
10'        ESEMPIO DI USO DI FILES INDICI (TRS80)
20'        ==========================================
30'
40'        1 RECORD LOGICO = 1 RECORD FISICO
50'        ALLOCAZIONE DINAMICA DEI RECORDS NEL FILE PRINCIPALE CON
60'        USO DI UNA BITMAP
70'
80         MXUN=50:LREC=254
90         DIM PT$(MXUN),KEY$(MXUN),BMAP$(LREC)
100        OPEN "R",#1,"FILE":OPEN "R",#2,"INDICEF1"
110        FIELD #1,12 AS N$,242 AS XXX$          'VEDERE DISEGNO'
120        FOR I=1 TO LREC:FIELD #1,(I-1) AS D$,1 AS BMAP$(I):NEXT
130        FOR I=1 TO MXUN:FIELD #2,5*(I-1) AS D2$,2 AS PT$(I),
                3 AS KEY$(I):NEXT
140'
150        INPUT"NOME ";NOM$:IF NOM$="FINE" THEN CLOSE #1,#2:STOP
160'
170        GOSUB 500:IF Q<>1 THEN GOTO 150 ELSE GOSUB 220:
              ON Q GOTO 180,200,150
180        PRINT N$:GOTO 150
190'
200        INPUT"NUOVO NOME OK?";R$:IF R$<>"OK" OR R$<>"SI" THEN 150
                  ELSE LSET N$=NOM$:PUT #1,NREC:GOSUB 430:GOTO 150
210' ---------------------------------------------------------------------
220        GET #2,IX                        'RICERCA CHIAVE'
230        FOR M=1 TO MXUN
240             IF CVI(PT$(M))=0 THEN FR=M:GOTO 270
250        NEXT M
260'
270        FOR M=1 TO MXUN                       '#1----> FILE PRINCIPALE
280             IF CVI(PT$(M))=0 THEN 390 'NREC--->INDIRIZZO RECORD
290             IF KKEY$<KEY$(M) THEN 390 'IX---->INDIRIZZO FILE INDICE
300             IF KKEY$=KEY$(M) THEN 340
310        NEXT M
320'
330        IF CVI(PT$(M))=0 THEN 390
340        NREC=CVI(PT$(M)):GET #1,NREC
350        IF NOM$<LEFT$(N$,LEN(NOM$)) THEN 390
360        IF LEFT$(N$,LEN(NOM$))=NOM$ THEN INSER=M:Q=1:RETURN
370        M=M+1:GOTO 330
380'
390        INSER=M:GOSUB 640:GET #1,NREC:Q=2          'CHIAVE SCONOSCIUTA'
400        IF CVI(PT$(MXUN-1))<>0 THEN PRINT "FILE INDICE COMPLETO":
                      Q=3:RETURN
410        RETURN
420' ---------------------------------------------------------------------
430        GET #1,1:LSET BMAP$(NREC)=CHR$(1):PUT #1,1 'AGGIUNTA CHIAVE
440        FOR U=FR-1 TO INSER STEP -1
441             LSET KEY$(U+1)=KEY$(U):LSET PT$(U+1)=PT$(U)
442        NEXT U
450        LSET PT$(INSER)=MKI$(NREC):LSET KEY$(INSER)=KKEY$
460        PUT #2,IX
470        RETURN
480' ---------------------------------------------------------------------
490'                                  CALCOLO INDIRIZZO INDICE
500        KKEY$=LEFT$(NOM$,3)
510        IX=ASC(LEFT$(NOM$,1))-64
520        IX=INT((IX+1)/2):Q=1:IF IX=1 OR IX>13 THEN Q=2
530        RETURN
531'
```

```
532'
533'
540' -------------------------------------------------------------------
534'
535'
536'
550'                                    SOPPRESSIONE RECORD
560      GET #1,1:LSET BMAP$(NREC)=CHR$(0):PUT #1,1
570      FOR U=INSER TO FR-1:LSET PT$(U)=PT$(U+1):
                   LSET KEY$(U)=KEY$(U+1):NEXT U
580      PUT #2,IX:RETURN
590' -------------------------------------------------------------------
630'                                    RICERCA TRAMITE LA BITMAP
640      GET #1,1
650      IF BMAP$(LREC)<>"*" THEN PRINT "ATTESA INIZIALIZZAZIONE":
                   GOSUB 740
660'
670      FOR W=2 TO LREC
680          IF ASC(BMAP$(W))=0 THEN NREC=W:RETURN
690      NEXT W
700' -------------------------------------------------------------------
710'                                    INIZIALIZZAZIONE
740      FOR P1=1 TO LREC:LSET BMAP$(P1)=CHR$(0):NEXT P1
750      LSET BMAP$(LREC)="*":PUT #1,1
755      FOR P2=1 TO 13
760          GET #2.P2:FOR Q1=1 TO MXUN:LSET PT$(Q1)=MKI$(0):
                   NEXT Q1:PUT #2.P2
770      NEXT P2
780      RETURN
790' -------------------------------------------------------------------
800'
810'                                    ESEMPIO
820'     NOM$="MARTINO"
830'     GOSUB 500:GOSUB 220:ON Q GOTO X,Y
840'     X -------------> CHIAVE TROVATA
850'
860'     Y -------------> GOSUB 430        (AGGIUNTA)
870'
880' -------------------------------------------------------------------
```

FILE INDICE
50 chiavi per record

FILE PRINCIPALE

CAPITOLO 6

DATA BASE

Spesso in ambito gestionale ci si trova di fronte a problemi che per la loro risoluzione pratica richiedono l'aggiornamento e la consultazione di archivi in tempo reale. Per poter affrontare in modo adeguato questo problema è necessario introdurre qualche concetto relativo all'uso del Data Base anche se attualmente non vi sono grosse applicazioni di questo tipo di strutturazione dei dati nel campo dei Personal Computers, in quanto il suo costo è ancora piuttosto elevato. Inoltre il Data Base risulta meno flessibile dei files (se questi sono indirizzati dinamicamente) anche se solleva il programmatore dal problema relativo alla gestione dei puntatori e propone un linguaggio standard.

Per questo, parlare di Data Base nel campo dei Personal Computers significa solo parlare di puntatori tra records di più files. Tali puntatori dovranno essere gestiti direttamente dal programmatore, ma vedremo comunque che tale gestione è più semplice del previsto.

Questo capitolo risulta diviso in quattro sezioni:

- Perchè i data base
- La gestione dei puntatori
- Controlli contro la distruzione di files
- Esempi di data base.

PERCHÈ IL DATA BASE

Nella nostra trattazione faremo riferimento ad un esempio concreto: supporremo

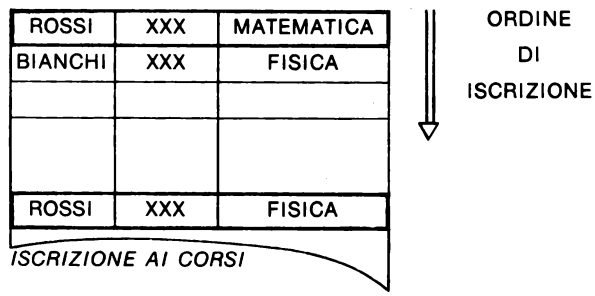
di voler 'gestire' l'insieme delle domande di iscrizione di studenti in vari corsi universitari. Immaginiamo a questo proposito che le domande vengano memorizzate in un file man mano che vengono presentate allo sportello *(figura 1).*

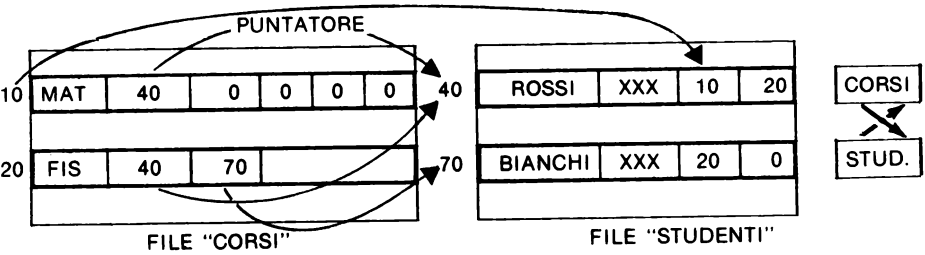
È evidente che utilizzando solo questa fonte di informazioni non è possibile verificare immediatamente, prima di accettare l'iscrizione, se vi sono ancora posti disponibili (per soli 1000 records infatti la ricerca sequenziale richiede già quasi un minuto).

Inoltre si può osservare che le notizie relative ad uno studente che vuole frequentare più corsi devono essere memorizzate più volte.

ROSSI	XXX	MATEMATICA
BIANCHI	XXX	FISICA
ROSSI	XXX	FISICA

ORDINE
DI
ISCRIZIONE

ISCRIZIONE AI CORSI

È logico a questo punto chiedersi se è possibile rendere 'indipendenti' le informazioni relative allo studente rispetto a quelle relative ai corsi.

Per risolvere questo problema si possono organizzare le informazioni in due files distinti (file dei corsi e file degli studenti) facendo in modo che i records dei due files siano correlati tra loro tramite dei puntatori *(figura 2).*

Vediamo ora come realizzare praticamente i legami logici esistenti tra i records dei due files.

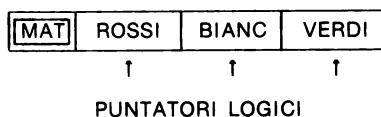
– In ogni record del file CORSI vengono memorizzati dei *puntatori* che 'puntano' ai

records del file STUDENTI in cui sono memorizzate le informazioni relative agli studenti iscritti a quel determinato corso.

— In modo analogo, in ogni record del file STUDENTI vengono memorizzati dei *puntatori* che 'puntano' ai records del file CORSI relativi ai corsi seguiti da quello studente.

Organizzando i dati in questo modo si può verificare, all'atto dell'iscrizione, se il nome dello studente è già stato registrato, se vi sono ancora posti disponibili per il corso da lui richiesto, il numero ed il tipo di corsi a cui è già iscritto. Inoltre si può ottenere facilmente, a richiesta, l'elenco dei partecipanti ad un certo corso.

Poichè un puntatore occupa poco spazio in memoria (2 bytes bastano per gestire circa 64000 valori diversi) non sorgono grossi problemi neppure con files di dimensioni notevoli.

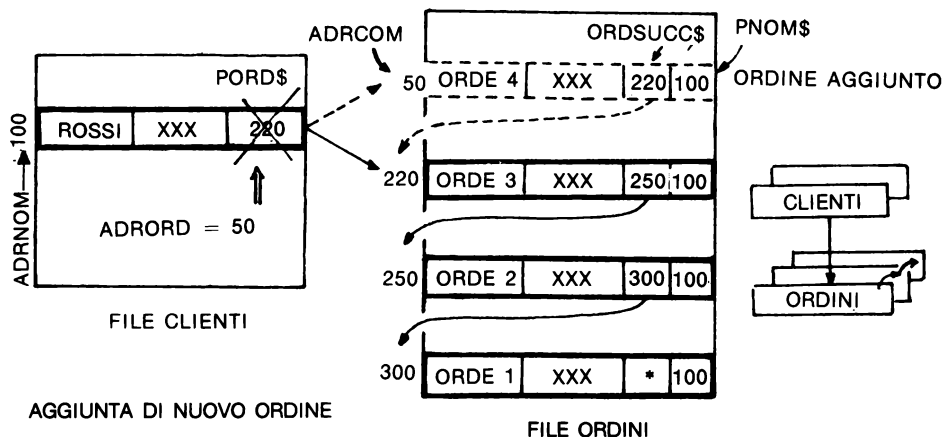
Rinunciando a questo vantaggio è possibile associare ad ogni record dei 'puntatori logici' (costituiti cioè da nomi simbolici) come illustrato nella *figura 3.* Adottando questa soluzione si rendono i due files più 'indipendenti': infatti la riorganizzazione di un file (ad es. CORSI) non implica necessariamente la riorganizzazione dell'altro. Per contro però lo spazio occupato dai puntatori risulta notevolmente aumentato (si passa da 2 a 5 o 10 bytes per puntatore) e pertanto se i files sono di notevoli dimensioni si deve ricorrere, per la loro gestione, ad una tabella di conversione (vedasi accesso indicizzato).

GESTIONE DEI PUNTATORI

Aggiunta di elementi in una lista concatenata

Facciamo ora riferimento ad una ditta che voglia gestire automaticamente gli ordini dei vari clienti.

Supponiamo che le informazioni siano state opportunamente memorizzate (con il metodo sopra esposto) in due files denominati CLIENTI e ORDINI.

Se ad ogni cliente corrisponde più di un ordine risulta indispensabile concatenare tra loro gli ordini relativi ad uno stesso cliente *(figura 4)*.

Quando giunge un nuovo ordine, questo deve ovviamente essere concatenato con quelli registrati in precedenza.

Per effettuare il concatenamento si portano in memoria i records interessati (uno del file CLIENTI ed uno del file ORDINI) e si procede all'aggiornamento dei puntatori ORDSUCC$, PNOM$, PORD$.

```
GET FILECLIENTI,ADRNOM          'LETTURA DAI FILES DEI DUE
GET FILEORDINI,ADRORD           'RECORDS INTERESSATI

-----------------------------

LSET ORDSUCC$=PORD$             'TRASFERISCE VECCHIO PUNTATORE
                                 IN ORDSUCC$
LSET PORD$=MKI$(ADRORD)         'TRASFERISCE INDIRIZZO DEL
                                 NUOVO ORDINE IN PORD$
LSET PNOM$=MKI$(ADRNOM)         'TRASFERISCE INDIRIZZO CLIENTE
                                 IN PNOM$

-----------------------------

PUT FILECLIENTI,ADRNOM          'AGGIORNAMENTO DEI
PUT FILEORDINI,ADRORD           'FILES
```

Nota: Quando aggiungendo un nuovo cliente nel file CLIENTI non si hanno ordini, il puntatore PORD$ viene inizializzato con

LSET PORD$ = '*'

L'inserimento di un nuovo elemento in una lista può essere fatto in modo da non alterare l'ordine della lista stessa.

Per avere la *lista* di tutti gli ordini di un cliente, di cui si conosce la posizione nel file CLIENTI (ADRNOM) si può utilizzare la seguente routine

```
        GOSUB LETTURA:STOP

LETTURA:GET FILECLIENTI,ADRNOM 'LETTURA RECORD DEL CLIENTE
        IF PORD$="*" THEN RETURN     'NON VI SONO ORDINI'
        ORD=CVI(PORD$)
CICL    :GET FILEORDINI,ORD:PRINT ORDINE$ 'ORDINE$:RECORD
                                           RELATIVO AD
                                           ORDINE
        IF ORDSUCC$="*" THEN RETURN
        ORD=CVI(ORDSUCC$):GOTO CICL
```

Esempio pratico: Sequenza di operazioni relative ad un ordine

Quando giunge un nuovo ordine, questo deve essere inserito nella lista degli ordini di quel determinato cliente: deve quindi essere 'concatenato' con gli ordini precedenti. Esplicitiamo i passi necessari per eseguire questa operazione.

Si esegue una ricerca sequenziale nel file CLIENTI per controllare se l'ordine è relativo ad un vecchio cliente oppure ad uno nuovo. Questa operazione viene effettuata mediante un sottoprogramma che, ricevuto il nominativo del cliente, ne ricerchi il corrispondente record nel file CLIENTI e nel caso questo non esista crei automaticamente un nuovo record (cliente nuovo) e inizializzi il relativo campo PORD$ con '*'. Come si vede questo sottoprogramma permette di aggiornare unicamente il file CLIENTI.

Il sottoprogramma sopra citato, che chiameremo CLIENTE, chiama a sua volta un altro sottoprogramma, che chiameremo CAL-ADRNOM, incaricato di ricercare il record relativo al nominativo richiesto (se tale record è presente nel file) oppure un record vuoto (cliente nuovo). Nel fare ciò questo secondo sottoprogramma posiziona un indicatore Q a 1 se il cliente è stato trovato oppure a 2 se il cliente è nuovo. Il valore di questo indicatore viene poi testato dal sottoprogramma CLIENTE per sapere come deve utilizzare i dati relativi al cliente in esame.

Alla fine del sottoprogramma CLIENTE comunque i dati relativi a quel cliente saranno memorizzati nel file. Per comunicare che non vi sono più dati da trasmettere l'utente dovrà rispondere alla domanda "Cliente?" con un CR. In questo caso il sottoprogramma CLIENTE posizionerà un indicatore Q a 2 e ripasserà il controllo al programma principale.

Avendo strutturato il sottoprogramma in questo modo, la routine per aggiungere un nuovo cliente all'elenco si riduce alle istruzioni:

```
CREACLIENT GOSUB CLIENTE:ON Q GOTO CREACLIENT,FINE
```

Sono stati poi previsti un sottoprogramma per l'aggiornamento dei dati relativi ad un cliente ed un altro sottoprogramma, detto CAL-ADRORD che ha il compito di trovare nel file ORDINI un record vuoto in cui registrare un nuovo ordine. Se per caso l'ordine in esame è già stato registrato in precedenza il sottoprogramma CAL-ADRORD posiziona un indicatore ad 1 in modo da evitare una registrazione superflua.

È stato previsto infine l'uso di un file ARTICOLI e di un sottoprogramma che controlli per ogni ordine se gli articoli richiesti sono disponibili, e se si tratta di un nuovo articolo ne generi automaticamente il record relativo.

Questo sottoprogramma può essere usato anche per aggiungere direttamente dei nuovi articoli nell'apposito file.

Osserviamo per concludere che prima di effettuare una registrazione in un file viene chiesta all'utente una conferma sui dati da registrare.

```
                         GESTIONE FILE ORDINI
                         ====================

INIZIO   :INPUT"TIPO OPERAZIONE ";MODE$
          IF MODE$="ORDINE" THEN GOTO ORDINE
          IF MODE$="         " THEN .......
               !
               !
               V

          GOTO INIZIO

ORDINE:GOSUB CLIENTE:ON Q GOTO 1,INIZIO

     1:INPUT"ORDINE";ORD$
       IF ORD$="" THEN GOTO ORDINE
       GOSUB CAL-ADRORD:ON Q GOTO 2,3

     2:PRINT"GIA' REGISTRATO":GOTO 1
     3:INPUT"DATI:";........
          :       :     :
          :       :     :
          :       :     :
 +-> 4:GOSUB ARTICOLI:ON Q GOTO 5,6
 |   5:INPUT   ..DATI..
 |      :        :
 |      :        :
 |      :        :
 +---  GOTO 4

     6:INPUT"OK?";RP$
       IF RP<>"SI" THEN GOTO ORDINE
```

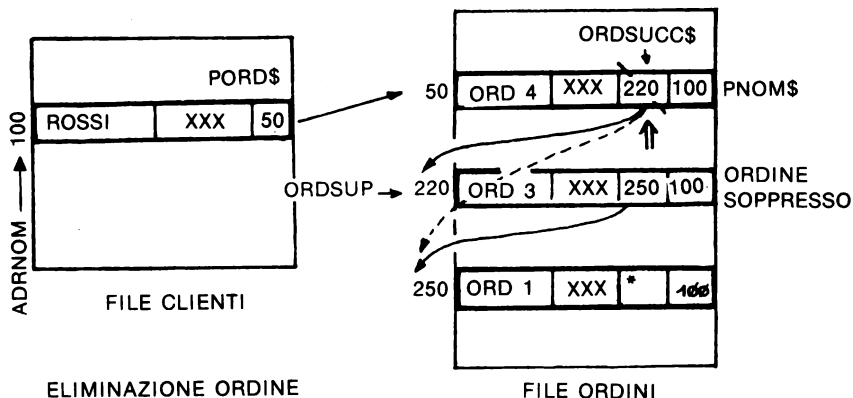
```
SOTTOPROGRAMMA PER RICERCA
NEL FILE "CLIENTI"

ALLA USCITA: Q=1   OK
             Q=2   FINE

CLIENTE:INPUT"NOME";CLIENT$
        IF CLIENT$="" THEN
           Q=2:RETURN
        GOSUB CAL-ADRNOM
        ON Q GOTO 1,2

      2:INPUT"N? ";RP$
        IF RP$<>"SI" THEN
           GOTO CLIENTE
        INPUT DATI ....
          :      :
          :      :
        -------------------
        LSET PORD$="*"     { INIZIALIZZAZIONE
        PUT .........      { PUNTATORE
        -------------------

      1:Q=1:RETURN
```

```
PUNTATORI( LSET ORDSUCC$=PORD$
         ( LSET PORD$=MKI$(ADRORD)
         ( LSET PNOM$=MKI$(ADRNOM)
-------------------------------------------
          PUT FILENOM,ADRNOM
          PUT FILEORD,ADRORD
-------------------------------------------


          GOTO ORDINE
===========================================

RICORDIAMO CHE L'INDICATORE Q VIENE
TESTATO ALL'USCITA DEI SOTTOPROGRAMMI
MEDIANTE UN'ISTRUZIONE DEL TIPO

ON Q GOTO XX,YY,...

===========================================
```

```
SOTTOPROGRAMMA PER RICERCA
DEGLI ARTICOLI

ALLA USCITA: Q=1   OK
             Q=2   FINE

ARTICOLI:INPUT ART$
         IF ART$="" THEN
         Q=2:RETURN

         GOSUB CAL-ADRAR
         ON Q GOTO 1,2

       2:INPUT"NUOVO?";R$
       IF R$<>"SI" THEN
         GOTO ARTICOLI

         INPUT ..DATI..

       1:Q=1:RETURN
```

Eliminazione di un elemento in una lista

Consideriamo il file ORDINI ed immaginiamo che un cliente disdica un ordine dalla lista di quelli relativi al cliente in questione. Per far ciò è sufficiente modificare il puntatore ORDSUCC$ contenuto nel record relativo all'ordine che precede nella lista quello da eliminare (vedi *figura 6*). Poichè gli elementi della lista non sono necessariamente contigui nel file ORDINI, sarà necessario ripercorrere, grazie ai puntatori, tutta la lista per ritrovare l'elemento che 'punta' a quello da eliminare. Il posizionamento all'inizio della lista viene ottenuto tramite il puntatore PORD$ contenuto nel record relativo al cliente (file CLIENTI). Osserviamo che tale record è facilmente rintracciabile grazie al puntatore PNOM$.

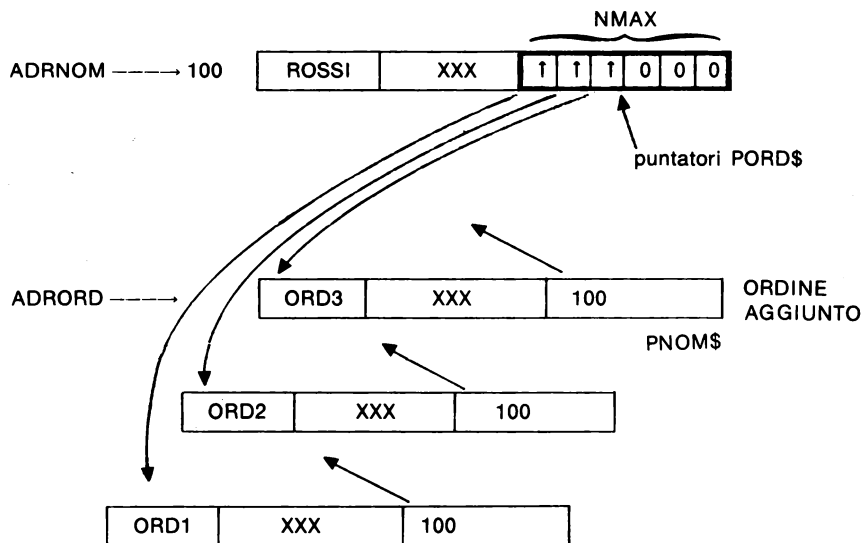
ELIMINAZIONE ORDINE

FILE ORDINI

```
      GOSUB SUP:STOP                      'SI TRASMETTE L"INDIRIZZO
                                           DELL"ORDINE DA SOPPRIMERE
      ADRNOM=CVI(PNOM$)
SUP   GET FILENOMI,ADRNOM
      IF PORD$"* " THEN PRINT"LISTA VUOTA":Q=3:RETURN
      IF CVI(PORD$)=ORDSUP THEN GET FILEORDINI,CVI(PORD$):
         LSET PORD$=ORDSUCC$:PUT FILENOMI,ADRNOM:Q=1:RETURN
                                  'ELIMINAZIONE PRIMO ORDINE'

      X=CVI(PORD$)                 'INDIRIZZO PRIMO ORDINE'

CICL  GET FILEORDINI,X
      IF CVI(ORDSUCC$)=ORDSUP THEN GET FILEORDINI,CVI(PORD$):
         Y$=ORDSUCC$:GET FILEORDINI,X:LSET ORDSUCC$=Y$:
         PUT FILEORDINI,X:Q=1:RETURN

      IF ORDSUCC$="* " THEN PRINT"ORDINE NON REGISTRATO":Q=2:RETURN

      X=CVI(ORDSUCC$):GOTO CICL
```

Qualora venga richiesta l'eliminazione di un ordine non esistente viene segnalato errore.

Una lista 'distrutta' può essere ricostruita andando a ricercare nel file ORDINI tutti i records in cui il puntatore PNOM$ ha il medesimo valore.

Osserviamo infine che le operazioni necessarie per riorganizzare una lista risultano notevolmente agevolate se si concatena la lista stessa nei due sensi (si può ad esempio individuare immediatamente l'elemento che precede un elemento dato).

Un altro tipo di organizzazione per il file CLIENTI è quello che consiste nel fissare all'inizio il numero massimo di ordini che possono comporre la lista di un cliente (nel nostro esempio 5). ed associare ad ogni 'cliente' un numero di puntatori pari al numero massimo di ordini accettabile *(figura 7)*. Con questo tipo di organizzazione si semplifica notevolmente la gestione dei puntatori e si facilitano pertanto le operazioni di inserimento e cancellazione.

AGGIUNTA DI UN ORDINE

Inserimento nuovo ordine

Per effettuare questo tipo di operazione devono essere presenti in memoria centrale i records di indirizzo ADRNOM ed ADRORD

```
        GOSUB INS:STOP

INS     FOR I=1 TO NMAX-1              'RICERCA PUNTATORE LIBERO'
             IF CVI(PORD$(I))=0 THEN
                  LSET PORD$(I)=MKI$(ADRORD):LSET PNOM$=MKI$(ADRNOM):
                  PUT FILECLIENTI,ADRNOM:PUT FILEORDINI,ADRORD:Q=1:
                  RETURN

        NEXT I
        PRINT"NON C'E' POSTO":Q=2:RETURN
```

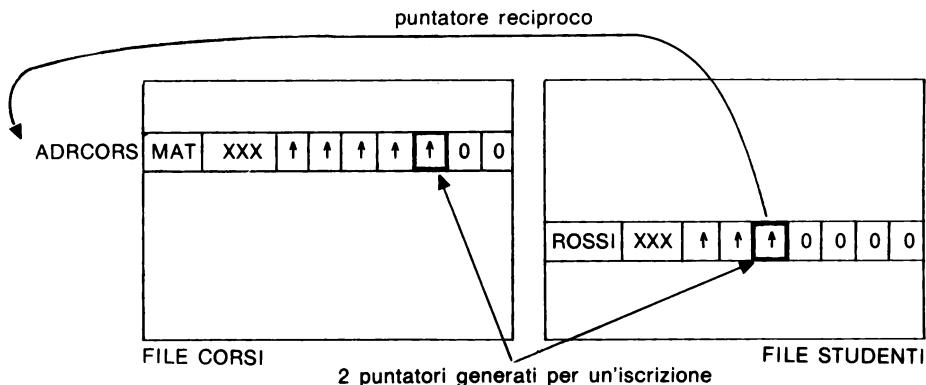
Soppressione ordine

Supponendo noto l'indirizzo dell'ordine da sopprimere, si legge il record del 'cliente' dal file CLIENTI e si azzera il puntatore relativo all'ordine da eliminare.

```
        GOSUB SUP:STOP

SUP     ADRNOM=CVI(PNOM$):GET FILECLIENTI,ADRNOM
        FOR I=1 TO NMAX-1
            IF CVI(PORD$(I))=ORDSUP THEN
                FOR V=I TO NMAX-1:LSET PORD$(I)=PORD$(I+1):NEXT V:
                PUT FILECLIENTI,ADRNOM:Q=1:RETURN

        NEXT I
        PRINT"ORDINE NON REGISTRATO":Q=2:RETURN
```

Se si desidera mantenere un ordinamento tra i vari ordini ricevuti è necessario "aggiustare" i puntatori in modo da non lasciare "buchi" all'interno della lista (figura seguente)

SOPPRESSIONE DI UN ORDINE

Controllo sui puntatori (figura 9)

Facciamo ora riferimento all'esempio illustrato all'inizio del presente capitolo e consideriamo i due file CORSI e STUDENTI.

Quando uno studente viene iscritto ad un corso vengono generati due puntatori:

1 nel file CORSI ——→ studente
1 nel file STUDENTI ——→ corso

Tramite questi puntatori è possibile ottenere facilmente un elenco di tutti gli studenti iscritti ad un corso ed un elenco dei corsi seguiti da uno studente.

Acquisiti i dati relativi ad un'iscrizione, si fanno due operazioni di scrittura: una sul file CORSI ed una sul file STUDENTI.

```
PUT FILECORSI,ADRCORS
                              <---- INTERRUZIONE
PUT FILESTUD,ADRSTUD
```

Se tra le due operazioni di scrittura si ha un'interruzione accidentale del programma, viene generato solo il primo puntatore. Per ridurre al massimo tale rischio conviene disporre queste due istruzioni una di seguito all'altra. Inoltre, al fine di evitare che, malgrado tutto, esistano delle incongruenze tra le varie liste di puntatori, conviene accertare, ad ogni accesso ad un record, che per ogni puntatore esista nell'altro file il puntatore inverso. Si tratta in pratica di andare ad esplorare il secondo file tramite il primo puntatore per vedere se nel record 'puntato' esiste un puntatore che richiama il record 'puntante'.

Se ad esempio siamo posizionati sul file CORSI e vogliamo leggere nel file STUDENTI i records relativi ai partecipanti ad un certo corso, dovremo verificare che in ognuno di questi records esista un puntatore che punta al record (del file CORSI) relativo al corso in oggetto.

```
        GOSUB VERIF:STOP

VERIF   FOR I=1 TO NMXPCORS
            IF CVI(PCORS$(I))=ADRCORS THEN Q=1:RETURN
        NEXT I
        PRINT"NON ESISTE PUNTATORE INVERSO":Q=2:RETURN
```

Se vi è incongruenza tra i puntatori, viene segnalata l'anomalia all'operatore. Si potrebbe anche predisporre una routine di autocorrezione per l'aggiunta del puntatore mancante.

L'uso dei puntatori reciproci permette inoltre di ricostruire una parte di record andata incidentalmente persa (purchè relativa ai soli puntatori).

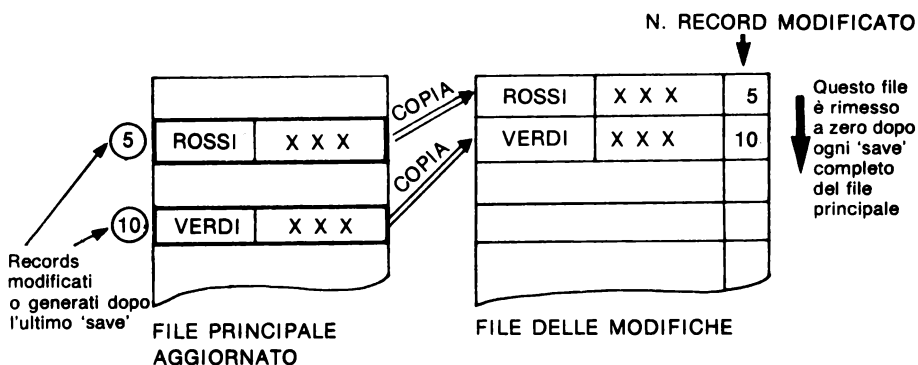
Protezione contro la distruzione fisica dei files
(figura 10)
Riportiamo una regola poco conosciuta.

Lavorando in real-time non si garantisce completamente la protezione di un file neppure effettuando delle operazioni di 'save' periodico su disco ad intervalli piutto-

sto brevi. Infatti in caso di interruzione vengono comunque perse tutte le modifiche effettuate nel file dopo l'ultimo 'save'. Conviene pertanto mantenere una traccia di queste modifiche (su un altro disco).

Con questo accorgimento se un file viene distrutto può essere rigenerato utilizzando l'ultima versione 'salvata' e la traccia delle modifiche non ancora registrate.

Quando anche il file delle modifiche raggiunge una certa mole, converrà effettuare un aggiornamento di tutti i files su disco in modo da poter riinizializzare il file delle modifiche.

Nota: Osserviamo che l'aggiornamento tramite il file delle modifiche è molto più veloce di quello ottenuto tramite un 'save' completo del file principale.

PROTEZIONE CONTRO LA DISTRUZIONE FISICA

ESEMPIO DI DATA BASE: GESTIONE DEGLI ORDINI CLIENTI/FORNITORI

Questo data base (illustrato in *figura 11*) deve permettere la gestione dell'archivio ordini clienti/fornitori. Si vogliono inoltre ricavare dai dati relativi al magazzino delle previsioni di massima per un certo periodo di tempo.

Quando deve essere accettato l'ordine di un cliente è necessario sapere se gli articoli richiesti saranno disponibili entro una certa data. Per ottenere queste informazioni bisogna esaminare per ogni articolo:

- la giacenza attuale
- gli ordini dei clienti
- gli ordini per i fornitori

DB CLIENTI FORNITORI
ORDINI - EVASIONI
ROSSI
INDIRIZZO
XXXX
ACCESSO DIRETTO
ARTICOLO Ⓨ
ARTICOLO Ⓧ
CODICE
PREZZO
C/F
STOCK INIZ.
STOCK ATTUALE
ACCESSO DIRETTO
C = CLIENTI
F = FORNITORI
ORD ①
ORD ②
ORD ③
C/F
XXX
QTA
PREZ-ZO
SET
QTA
PREZ-ZO
SET
XXXX
ARTICOLO ①
ARTICOLO ②
ACCESSO DIRETTO
SEM ①
SEM ②
SEM ㊿
ORDINI
CLIENTI
FORNIT.
10
15
7
8
4
20
EV ①
EV ②
XXXX
QTA EV
QTA EV
ARTICOLO ①
ARTICOLO ②
ACCESSO DIRETTO
CLIENTI/FORNITORI
ORDINI C/F
EVASIONI C/F
ARTICOLI
ACCESSO DIRETTO

L'aggiornamento dei dati in seguito all'evasione, anche parziale, di un ordine deve avvenire in tempo reale.

Dobbiamo pertanto organizzare il data base mediante dei files (clienti, ordini, consegne, articoli). Accedendo alla base tramite, ad esempio, i dati relativi ad una consegna, si potrà risalire, tramite appositi puntatori, all'indice ed al cliente a cui tale consegna si riferisce.

La base dei dati viene generata dinamicamente: ad esempio, al momento dell'accettazione di un ordine (da parte di un fornitore), se in esso è contenuto un nuovo articolo si potrà aggiornare il file ARTICOLI tramite un apposito sottoprogramma. La medesima cosa può essere fatta per il file CLIENTI ed il file FORNITORI.

Gestione degli ordini dei clienti

Il programma da noi sviluppato prevede la possibilità di accedere ai files solo tramite il nome del cliente e di gestire successivamente dei puntatori che concatenino gli ordini del cliente stesso. Pertanto, poichè non è previsto l'accesso al data base tramite il numero di un ordine, se si desiderasse poter effettuare anche questo tipo di operazione occorrerebbe modificare il programma in tale senso.

```
10      PRINT "DATA BASE":PRINT
20'
30'     GESTIONE ORDINI CLIENTI
40'     ===========================
50'
60      OPEN "R",#1,"CLIENTI":OPEN "R",#2,"ORDINI"
70      FIELD #1,20 AS N$,45 AS INDIR$,20 AS TL$,2 AS PORD$
80      FIELD #2,12 AS CORD$,50 AS DESC$,2 AS ORDSUCC$,2 AS PNOM$
90'
100     DIM NOM$(200),INDEX(200)
110     GOSUB 850                    'GENERAZIONE TABELLA INDICE/CLIENTE'
120'
130     PRINT:INPUT"OPERAZIONE(NUOVO,LISTA,FINE)";MODE$
140     IF MODE$="NUOVO" THEN 200 'NUOVO ORDINE'
150     IF MODE$="LISTA" THEN 710 'LISTA ORDINI'
160     IF MODE$="FINE" THEN CLOSE #1,#2:STOP
170     GOTO 130
180'---------------------------------------------------------------------
190'                                                  NUOVO ORDINE
200     GOSUB 380:ON Q GOTO 210,130    'RICERCA CLIENTE'
210     PRINT:PRINT N$:PRINT:NORD=LOF(2):PRINT"N.ORDINE ";NORD
220     GET #2,NORD
230'
240     PRINT:PRINT"ACQUISIZIONE ORDINE":PRINT
250     PRINT:INPUT"CODICE ORDINE ";X$:LSET CORD$=X$
260'
270'    .....ACQUISIZIONE ALTRI DATI RELATIVI ALL"ORDINE,
280'         E VISUALIZZAZIONE DEL RECORD COMPLETATO .....
290'
300     PRINT:INPUT"TUTTO OK?";R$:IF R$<>"SI" THEN GOTO 200
```

```
310    LSET ORDSUCC$=PORD$:LSET PNOM$=MKI$(ADRNOM)   'GENERAZIONE DEI
320    LSET PORD$=MKI$(NORD)                         'PUNTATORI
330'                                                 (CONCATENAZIONE)
340    PUT #1,ADRNOM:PUT #2,NORD
350    GOTO 200
360'------------------------------------------------------------------
370'                               RICERCA/GENERAZIONE CLIENTE
380    PRINT:INPUT"CLIENTE ";X$
390    IF X$="" THEN Q=2:RETURN                      'FINE OPERAZIONE'
400'
410    L=LEN(X$)
420'
430    FOR I=1 TO 200
440      IF X$=LEFT$(NOM$(I),L) THEN ADRNOM=INDEX(I):GET #1,INDEX(I):
                Q=1:RETURN
450      IF NOM$(I)="" THEN 480
460    NEXT I
470'
480    PRINT:PRINT X$;:INPUT"NUOVO CLIENTE? ";RP$
490    IF RP$<>"SI" THEN 380
500'
510    ADRNOM=LOF(1):GET #1,ADRNOM:LSET N$=X$:PRINT
                               'AGGIUNTA ALLA FINE DEL FILE'
520    INPUT"INDIRIZZO ";X$:LSET INDIR$=X$     'ACQUISIZIONE DATI'
530    INPUT"TELEFONO ";X$:LSET TL$=X$
560'
570    LSET PORD$="*"                        'INIZIALIZZAZIONE PUNT.
580    PUT #1,ADRNOM
590    NOM$(PI)=N$:INDEX(PI)=ADRNOM:PI=PI+1  'AGGIORNAMENTO TABELLA
610    Q=1:RETURN
620'------------------------------------------------------------------
630'
640'
650'
700'
710    GOSUB 380:ON Q GOTO 720,130                'LISTA ORDINI
720    PRINT:PRINT N$:PRINT                       'DI UN CLIENTE
730    PRINT TAB(20) INDIR$
740'
770    PRINT:IF PORD$="*".THEN GOTO 170           'NESSUN ORDINE
780    ORD=CVI(PORD$)
790'
800    GET #2,ORD:PRINT"ORDINE ";CORD$;"          ";DESC$
810    IF ORDSUCC$="*" THEN GOTO 710              'ULTIMO ORDINE
820    ORD=CVI(ORDSUCC$):GOTO 800                'ORDINE SUCCESSIVO
830'
840                               'GENERAZIONE TABELLA NOME/INDICE
850    PI=1
860                               'PI: PUNTATORE ALLA TABELLA
870    FOR I=1 TO LOF(1)
880        GET #1,I               'LETTURA SEQUENZIALE DEL FILE
890        IF ASC(N$)=0 THEN 910 'RECORD VUOTO
900        NOM$(PI)=N$:INDEX(PI)=I:PI=PI+1
910    NEXT I
920'
930    RETURN
940'
950'------------------------------------------------------------------
```

APPENDICE I

RICHIAMI SUL BASIC

Attivazione del Basic

L'attivazione dell'interprete Basic viene ottenuta tramite un comando da consolle:

BASIC (return per TRS80 e DTC MICROFILE

o

MBASIC (return) per BASIC MICROSOFT 5.

Dopo l'inizializzazione il sistema BASIC risponde:

OK (OK permette di assicurarsi che si è sotto il controllo del BASIC)
(Sul TRS80 il sistema risponde READY)

L'utente può a questo punto utilizzare il BASIC per:

- fare direttamente dei calcoli (Direct mode)
- scrivere dei programmi (Program mode)

Direct Mode

Battendo sulla tastiera

PRINT 4 (return)

il calcolatore visualizza il numero 4

Introducendo successivamente

A = 4 (return)
B = 5 (return)

il calcolatore assegna alle variabili A e B i valori 4 e 5.
Se a questo punto battiamo

```
PRINT A+B        (return)
```

il calcolatore visualizza 9 (somma dei valori contenuti in A e B)

Program mode

Scriviamo ora il seguente programma:

```
10  A=4
20  B=5
30  PRINT A+B
```

In esso 10, 20, 30 rappresentano dei numeri di istruzione.

Quando chiediamo al calcolatore di eseguire il programma, tramite il comando 'RUN', le istruzioni vengono eseguite l'una dopo l'altra secondo l'ordine di numerazione.

```
RUN        (return)
```

9 viene visualizzato durante l'esecuzione

Un programma può essere rieseguito tutte le volte che si desidera.
Per registrare su disco il programma si userà il comando

```
SAVE "PIPPO"
```

(ove PIPPO è il NOME assegnato al PROGRAMMA)

Per ricaricare il programma in memoria centrale (per poterlo eseguire) si ricorrerà al comando:

```
LOAD "PIPPO"
```

Dopo l'esecuzione di un programma si possono utilizzare i valori assegnati alle variabili passando in direct mode.

Se ad esempio, facendo riferimento al programma precedente, si batterà alla fine dell'esecuzione,

```
PRINT A*B
```

il calcolatore visualizzerà il numero 20.

Somma di due numeri

Generalizzando il programma precedente, immaginiamo di voler fare la somma di due numeri qualsiasi: in questo caso il programma dovrà chiedere all'operatore, durante l'esecuzione, il valore dei due numeri da addizionare.

Quando viene eseguita l'istruzione

```
10 INPUT "PRIMO NUMERO";A
```

viene visualizzato il messaggio "PRIMO NUMERO" da parte del programma che rimane poi in attesa dell'inserimento del dato.

L'istruzione 20 viene eseguita in modo analogo.

L'istruzione

```
30 PRINT "SOMMA=";A+B
```

permette di visualizzare il messaggio SOMMA=seguito dal valore di A+B

Infine l'istruzione

```
40 GOTO 10        (vai all'istruzione 10)
```

provoca nel programma un salto all'istruzione 10 che verrà nuovamente eseguita.

Tutto ciò che in un programma compare tra apici (') e segue su una riga la parola REM (REMARK) va considerato come commento.

```
LOAD "EX1"          <------------------COMANDO PER CARICARE IL PROGRAMMA
OK
LIST                <------------------COMANDO PER VISUALIZZARE IL LISTING

2 REM   SOMMA DI 2 NUMERI
4 REM   ==================
6'
10 ┌→  INPUT"PRIMO NUMERO ";A        'ACQUISIZIONE PRIMO VALORE'
20 │   INPUT"SECONDO NUMERO ";B      'ACQUISIZIONE SECONDO VALORE'
30 │   PRINT "SOMMA = ";A+B          'VISUALIZZAZIONE DELLA SOMMA
40 └─  GOTO 10                       'SALTO ALL"ISTRUZIONE 10'

RUN                 <------------------RICHIESTA DI ESECUZIONE

PRIMO NUMERO 60
SECONDO NUMERO 40
SOMMA = 100
PRIMO NUMERO 70
SECONDO NUMERO 50
SOMMA = 120
PRIMO NUMERO        <-----PER INTERROMPERE IL PROGRAMMA BATTERE IL
                          TASTO BREAK OPPURE CONTROL C
```

```
     ----------------->!
    !                  !
    !          ------------------
    !         / RICEVI    A    /           10
    !         ----------------
    !                  !
    !          ------------------
    !         / RICEVI    B    /           20
    !         ----------------
    !                  !
    !          ------------------
    !         / STAMPA  A+B    /           30
    !         ----------------
    !                  !
     ---------<-----------

               FLOW  CHART
               -----------
```

Somma di 10 numeri

(concetto di contatore)

Per sommare più numeri (ad esempio 10) si potrebbe procedere riscrivendo per 10 volte le stesse istruzioni, come illustrato nell'esempio precedente.

È però più conveniente utilizzare una serie di istruzioni 'cicliche' definendo un 'contatore' che permetta di contare il numero di volte che viene ripetuta la serie di i-struzioni.

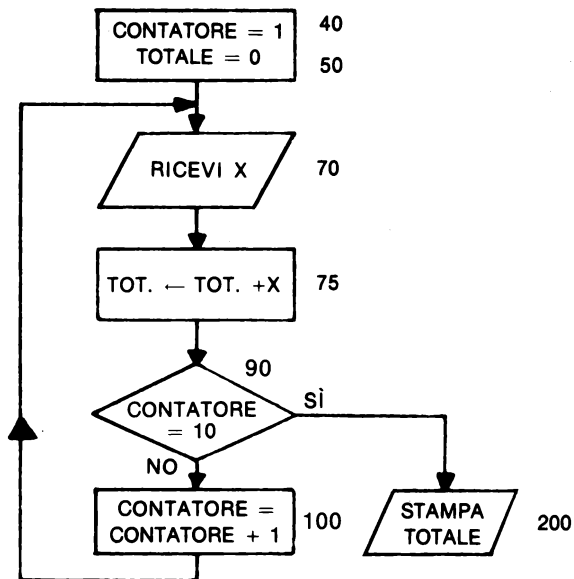
Per 'uscire' dal ciclo si utilizzerà un test sul valore assunto dal contatore in modo che il ciclo termini quando il contatore assume il valore 10.

L'istruzione: 90 IF CONTATORE=10 THEN 200 (se CONTATORE=10 salta alla istruzione 200)

fa si che quando il contatore assume il valore 10 il programma prosegua con l'istruzione 200. Se invece il valore del CONTATORE è diverso da 10 il programma prosegue con l'esecuzione dell'istruzione 100.

Nota: nell'esempio viene presentato il comando 'TRON' (Trace On) molto utile in fase di messa a punto. Tramite tale comando è possibile verificare come in realtà 'gira' il programma. Se si vuole disattivare l'effetto di TRON basta usare il comando 'TROFF'.

```
 10'       SOMMA DI 10 NUMERI
 20'       ==========================
 30'
 40        CONTATORE=1
 50        TOTALE=0
 60'
 70        INPUT"NUMERO ";X
 80        TOTALE=TOTALE+X
 90        IF CONTATORE=10 THEN 200    'CONTROLLO SUL CONTATORE'
 100       CONTATORE=CONTATORE+1
 110       GOTO 70                     'RITORNO ALL"INIZIO DEL CICLO
 120'
 200       PRINT"TOTALE = ";TOTALE     'STAMPA DELLA SOMMA'

OK
RUN

NUMERO 4
NUMERO 6
NUMERO 7
NUMERO 5
NUMERO 3
NUMERO 5
NUMERO 9
NUMERO 8
NUMERO 5
NUMERO 6
TOTALE = 58

OK
TRON                    <--------------- TRACE ON
OK

[10][20][30][40][50][60][70]NUMERO 4
[80][90][100][110][70]NUMERO 6
  :                  :
  :                  :
[80][90][100][110][70]NUMERO 5
[80][90][100][110][70]NUMERO 6
[80][90][200]TOTALE = 58

OK
```

Nota: Sul TRS80 cambiare TOTALE con SOMMA

Vettori

Utilizzando il metodo precedente per calcolare la somma di 10 numeri, il valore di tali numeri non viene mantenuto in memoria e pertanto non è possibile effettuare su di essi altre elaborazioni. Per facilitare l'elaborazione su un numero considerevole di dati è conveniente memorizzarli/sotto forma di vettore.

Un vettore non è altro che un insieme di elementi ordinati a cui si può accedere nel seguente modo:

– se si vuole assegnare il valore 15 al terzo elemento del vettore A si utilizza l'istruzione

A(3)=15

Generalizzando il discorso, si può dire che l'accesso ad un vettore viene realizzato tramite un indice posizionale (I).

```
                  VETTORE    A
                 -------------
      A(1)       :    10     :
                 -------------
      A(2)       :    20     :
                 -------------
      A(3)       :    15     :
                 -------------
      A(4)       :    12     :
                 -------------
       :                 :
       :                 :
```

```
   10        I=1
┌→ 20        INPUT"NUMERO";X
│  25        A(I)=X
│  30        IF I=5 THEN 100
│  40        I=I+1
└─ 50        GOTO 20
   :          :
   100'      PROSECUZIONE PROGRAMMA
```

La prima volta I ha il valore 1 e l'istruzione 25 è equivalente a:

A(1)=X

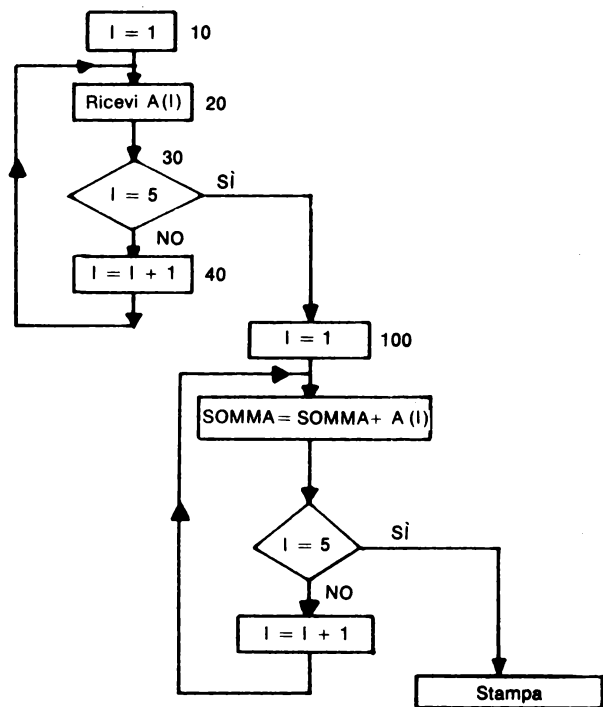
Con questa istruzione si assegna al primo elemento del vettore A il valore di X.

Giunti all'istruzione 40 l'indice I viene incrementato di 1, assumendo così il valore 2. Nel successivo passaggio l'istruzione 25 sarà quindi equivalente a:

A(2)=X

ecc.

Viene ora presentato come esempio il programma per calcolare la somma degli elementi di un vettore A.

```
2'          SOMMA DEGLI ELEMENTI DI UN VETTORE
4'          ==================================
6'
8'
10          I=1
20          INPUT"NUMERO ";A(I)               INIZIALIZZAZIONE VETTORE'
30          IF I=5 THEN GOTO 100
40          I=I+1:GOTO 20
45'
100         I=1
110         SOMMA=SOMMA+A(I)
120         IF I=5 THEN GOTO 300
130         I=I+1:GOTO 110
140'
300         PRINT"SOMMA = ";SOMMA
310         PRINT"MEDIA = ";SOMMA/5
320         STOP

OK
RUN

NUMERO 4
NUMERO 8
NUMERO 7
NUMERO 6
NUMERO 2
SOMMA = 27
MEDIA = 5.4

OK
PRINT A(3),A(5)          <---------------- DIRECT MODE
7            2

OK
PRINT A(3)+A(5)          <---------------- DIRECT MODE
9

OK
PRINT SOMMA/5            <---------------- DIRECT MODE
5.4

OK
```

Cicli FOR

Il programma precedente poteva essere scritto anche nel modo seguente:

```
 10        FOR I=1 TO 5                 'PER I CHE VA DA 1 A 5
→20             INPUT"NUMERO";X
 25             A(I)=X
←30        NEXT I                       'I=I+1 E VAI A 20 SE I<=5
```

Le due scritture sono equivalenti, ma questa seconda forma è più concisa e facilita la comprensione del programma.

In questo ciclo FOR tutte le istruzioni comprese tra FOR e NEXT vengono eseguite una prima volta con I=1 e poi ripetute con I=2,3,..., finchè I assume il valore limite specificato (nel nostro caso 5). A questo punto il programma prosegue con l'esecuzione dell'istruzione successiva al NEXT.

In 'uscita' dal ciclo, I assume il valore limite + 1 (nel nostro caso I=6).

Altri esempi di ciclo FOR

Tavola dei quadrati

```
  10        FOR I=1 TO 10
┌►20            PRINT I,I*I
└ 30        NEXT I

 RUN

            1         1
            2         4
            3         9
            4         16
            5         25
            :         :
            10        100
```

Tabella di moltiplicazione

```
 10        INPUT"RELATIVA A QUALE VALORE? ";X%
 20'
┌►30        FOR I%=1 TO 10
│ 40            PRINT X%;"*";I%;"=";(X%+I%)
└ 50        NEXT I
 60'
 70        GOTO 10

 RUN

           RELATIVA A QUALE VALORE? 7
           7*1=7
           7*2=14
              :  :
              :  :
           7*10=70
           RELATIVA A QUALE VALORE?

 BREAK IN 10
```

Ordinamento degli elementi di un vettore

Affrontiamo ora il problema relativo all'ordinamento degli elementi di un vettore presentando una soluzione classica:

. si confronta il secondo elemento con il primo:
— se è maggiore l'ordine non viene alterato;
— se è minore si effettua un'inversione in modo da avere i due elementi ordinati:

```
110 IF A(I+1) < A(I) THEN SWAP A(I+1),A(I):INVERSIONE=1
    !                 !       !
    (Se A(I+1) < A(I) scambiali tra loro e poni INVERSIONE=1)
```

Si incrementa di 1 l'indice I in modo da effettuare il confronto tra il secondo ed il terzo elemento e, se necessario, si scambiano tra loro

e così via

Giunti così alla fine del vettore si controlla il valore della variabile INVERSIONE:

— se non vi sono stati scambi si è giunti alla fine della procedura (il vettore è ordinato);

— se invece vi è stato anche un solo scambio ci si riposiziona all'inizio del vettore per effettuare una nuova serie di confronti.

Alla fine della prima 'passata' l'elemento di valore maggiore si troverà sicuramente all'ultimo posto nel vettore.
Alla fine della seconda 'passata' l'elemento maggiore tra i rimanenti occuperà il penultimo posto.
...

Operando in questo modo, dopo diversi cicli (al più N) si otterrà l'ordinamento completo del vettore.

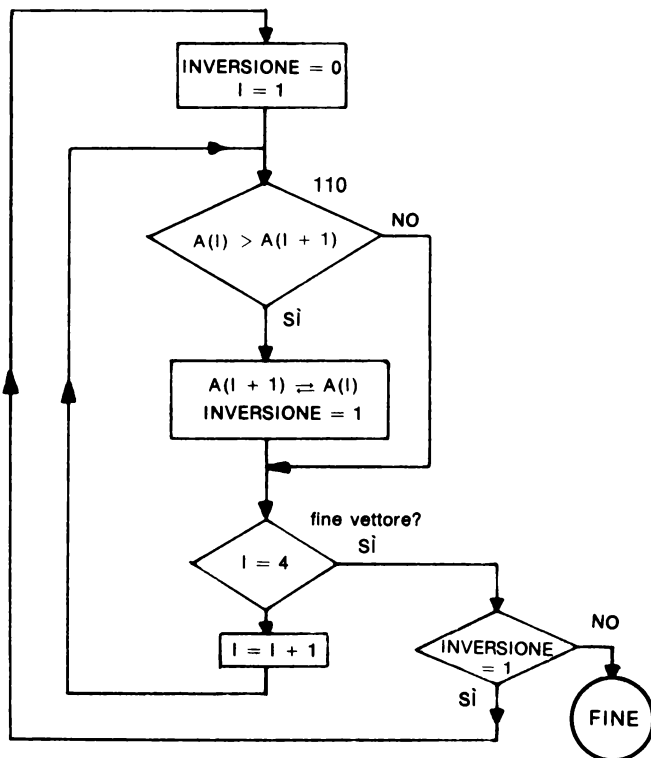
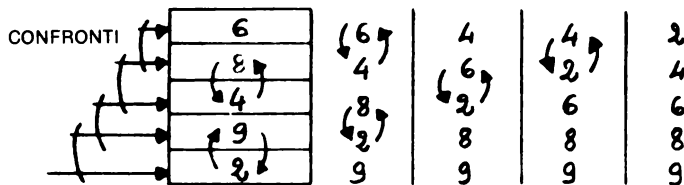
Il metodo utilizzato nell'esempio potrebbe essere migliorato perchè, per quanto sopra detto, si potrebbe ridurre di 1, ad ogni passaggio, il numero degli elementi da confrontare tra loro.

La variabile INVERSIONE viene utilizzata solo come variabile di controllo sul numero di scambi effettuati in una 'passata'.

Qualora il BASIC implementato sul vostro calcolatore non ammetta l'istruzione SWAP, potrete realizzare lo scambio tra due elementi nel modo seguente:

X=A(I):A(I)=A(I+1):A(I+1)=X

Osservazione: Il metodo da noi proposto (ripple sort) ha il vantaggio di essere concettualmente molto semplice e facile da programmare, ma in pratica viene raramente utilizzato, specie per vettori di notevoli dimensioni, in quanto è tra i più lenti che si conoscano.

```
10 REM    ORDINAMENTO DEI VALORI CONTENUTI IN UN VETTORE
20'       ==============================================
30'
40        FOR I=1 TO 5                   'ACQUISIZIONE DATI'
50             INPUT"NUMERO: ";A(I)
60        NEXT I
70' ------------------------------------------------------
80        INVERSIONE=0                   'ORDINAMENTO'
100       FOR I=1 TO 4
110          IF A(I+1)<A(I) THEN SWAP A(I+1),A(I):INVERSIONE=1
120       NEXT I
130'
140       IF INVERSIONE=1 THEN GOTO 80    'NON SI E" OTTENUTO UN
150                                       'ORDINAMENTO TOTALE'
160' ---------------------------------------------------------
170       PRINT"RISULTATO":PRINT          'VISUALIZZAZIONE VETTORE
180                                       'ORDINATO'
190       FOR I-1 TO 5
200            PRINT I,A(I)
210       NEXT I

RUN

NUMERO: 6
NUMERO: 8
NUMERO: 4
NUMERO: 9
NUMERO: 2

RISULTATO

1          2
2          4
3          6
4          8
5          9

OK

SE L'ISTRUZUONE SWAP NON E' AMMESSA SOSTITUIRLA CON:

          X=A(I):A(I)=A(I+1):A(I+1)=X
```

Nell'esempio precedente si è considerato un vettore di 5 elementi; per adeguare il programma ad un vettore di NB elementi occorrerà aggiungere le seguenti istruzioni:

```
20        INPUT"NUMERO DI ELEMENTI ";NB
30        DIM A(NB)
```

L'istruzione 30 permette di riservare in memoria centrale un'area sufficiente per contenere un vettore di NB elementi. Questa istruzione non è presente nel programma precedente in quanto per vettori con un numero di elementi inferiore a 10 il dimensionamento viene fatto in modo automatico.

Nel programma inoltre va sostituito il parametro 5 con NB, che rappresenta il generico numero di elementi. Così, ad esempio, il ciclo di lettura dovrà essere trasformato nel modo seguente:

```
FOR I=1 TO NB
    INPUT"NUMERO ";A(I)
NEXT I
```

Ordinamento di nomi

Nell'esempio precedente si è fatto riferimento a variabili di tipo numerico. In generale, però, specie in ambito gestionale, si deve affrontare il problema relativo all'ordinamento alfabetico di stringhe di caratteri (ad esempio i nominativi dei clienti).

Per distinguere le variabili destinate a contenere stringhe di caratteri (che chiameremo d'ora in poi di tipo alfanumerico) da quelle destinate a contenere solo valori numerici si utilizza il carattere speciale $ posto subito dopo il nome della variabile stessa: così, se ad esempio il vettore A fosse di tipo alfanumerico, lo si dovrebbe indicare con la notazione A$.

Per confrontare tra loro due stringhe, supposto valido l'ordinamento alfabetico, si utilizza lo stesso tipo di istruzione valida per i dati numerici:

```
IF A$(I+I) < A$(I)THEN...
```

Il programma per l'ordinamento di una serie di nomi si differenzia da quello da noi presentato solo nei nomi da assegnare alle variabili (A$ al posto di A).

```
10 REM   ORDINAMENTO DI NOMI
20'      ====================
30'
32'
34       INPUT"NUMERO DI ELEMENTI DA ORDINARE ";NB
36       DIM A$(NB)
38'
40       FOR I=1 TO NB                'ACQUISIZIONE DATI'
50           INPUT"NOME ";A$(I)
60       NEXT I
70' ---------------------------------------------------------------
80       INVERSIONE=0                 'ORDINAMENTO'
90'                                    -----------
```

```
100        FOR I=1 TO NB-1
110            IF A$(I+1)<A$(I) THEN SWAP A$(I+1),A(I):INVERSIONE=1
120        NEXT I
130'
140        IF INVERSIONE=1 THEN GOTO 80
150'
160' -------------------------------------------------------------
170        PRINT"ELENCO ORDINATO":PRINT    'STAMPA ELENCO'
180'
190        FOR I=1 TO NB
200            PRINT I,A$(I)
210        NEXT I
```

```
RUN

NUMERO DI ELEMENTI DA ORDINARE 6

NOME ROSSI MARIO
NOME VERDI GIUSEPPE
NOME BIANCHI ARTURO
NOME ESPOSITO SALVATORE
NOME BELLINI SILVIA
NOME BELLANI ANTONIO

ELENCO ORDINATO

1          BELLANI ANTONIO
2          BELLINI SILVIA
3          BIANCHI ARTURO
4          ESPOSITO SALVATORE
5          VERDI GIUSEPPE
```

```
10'        ISTOGRAMMA
20'        ==========
30'
40'        VIENE GENERATO, A PARTIRE DA UN VETTORE A, UN VETTORE HIST
50'        CONTENENTE:
60'        COME PRIMO ELEMENTO IL NUMERO DI 1 PRESENTI NEL VETTORE A
70'        COME SECONDO ELEMENTO IL NUMERO DI 2 PRESENTI NEL VETTORE A
80'        :::::::
90'
100'       VETTORE A                          VETTORE HIST
110'       ----------
120'       :   3   :
130'       ----------
140'       :   2   :
150'       ----------                          ----------------
160'       :   3   :                  1       :    2       :
170'       ----------                          ----------------
           :   1   :                  2       :    3       :
180'       ----------      ====>               ----------------
190'       :   2     :                3       :    2       :
200'       ----------                          ----------------
210'       :   1   :                  4       :    1       :
240'
250        DIM A(20),IST(10)
260'
270        FOR I=1 TO 20                 'INIZIALIZZAZIONE DI A'
280           INPUT"NUMERO INTERO COMPRESO TRA 1 E 10 ";X
```

```
290         IF X>10 OR X<1 THEN PRINT "ERRORE:NUMERO NON ACCETTABILE":
                      GOTO 280
300         A(I)=X
310      NEXT I
320' ------------------------------------------------------------
330      FOR I=1 TO 20              'GENERAZIONE VETTORE HIST'
340         X=A(I)
350         HIST(X)=HIST(X)+1
360      NEXT I
370' ------------------------------------------------------------
380      FOR I=1 TO 10              'STAMPA DELL"ISTOGRAMMA'
410         LPRINT I;
420         IF HIST(I)=0 THEN GOTO 480
430         FOR J=1 TO HIST(I)
450               LPRINT"*";       ''USO DELLA STAMPANTE'
460         NEXT J
470'
480         LPRINT""
490      NEXT I
500' ------------------------------------------------------------
510'                                ESEMPIO
520'     1 **
530'     2 ***
540'     3 *
550'     4 **
560'     :::::::
600' ------------------------------------------------------------
```

Consigli pratici (importante)

Gli interpreti BASIC (e molto verosimilmente anche il vostro) offrono delle facilitazioni per la messa a punto dei programmi, che converrà tener presente.

– Si può interrompere un programma mediante un BREAK (o un Control C) ed andare a testare, in 'direct mode', il contenuto delle variabili per controllare l'EVOLUZIONE dei loro valori durante l'esecuzione del programma.

Per esempio:

    PRINT CONTATORE

Si può poi riprendere l'esecuzione del programma tramite un CONT (CONTINUE).

Il 'direct mode', oltre al controllo, permette anche di modificare direttamente il valore delle variabili.

Ad esempio:

CONTATORE=8	assegna il valore 8 alla variabile CONTATORE
CONT	fa riprendere l'esecuzione

L'esecuzione di un programma, dopo un'interruzione, può essere ripresa da un qualsiasi punto utilizzando, in 'direct mode', il comando:

GOTO xx	ove xx rappresenta il numero di linea da cui si vuole riprendere l'esecuzione.

Con questo accorgimento si evita, in fase di messa a punto, di dover ripetere sempre anche la parte di programma già controllata. Inoltre utilizzando il comando GOTO xx non si ha la riinizializzazione a zero di tutte le variabili, come avviene invece utilizzando il comando RUN.

Quasi tutte le istruzioni, comprese quelle per la gestione dei files, sono utilizzabili come comandi in 'direct mode'.

APPENDICE II

MESSAGGI DI ERRORE DEL BASIC MICROSOFT

Se il messaggio di errore non è sufficiente per risalire alla causa che lo ha generato, converrà controllare in 'direct mode' il contenuto delle variabili per facilitare la diagnosi dell'errore.

Se poi è necessario controllare in modo più completo l'evolvere del programma, converrà inserire nel programma stesso, dislocandoli opportunamente, degli STOP che interrompano il programma al momento più opportuno.

Si può infine ricorrere all'inserimento di alcune istruzioni di 'stampa' che permettano di controllare costantemente il contenuto di alcune variabili. (Un PRINT X, ad esempio, permette di controllare il valore della variabile X).

L'uso della 'traccia' (cfr TRON) è conveniente solo in casi molto complicati, dato che fornisce un numero di informazioni spesso troppo elevato.

Messaggio	**Codice**	
Bad file mode	54	(Errato uso di un file) È stato usato un PUT o un GET per un file di tipo sequenziale oppure è stato usato in un OPEN un codice diverso da R,0,I
Bad file number	52	(Errato numero di file) Si vuole eseguire un'operazione su un file con un numero di identificazione sconosciuto.
Bad file name	64	(Errato nome del file) Il nome che si vuole assegnare ad un file non è accettabile (ad esempio troppi caratteri).

Bad record number	63	(Errato numero di record) In un'istruzione PUT o GET viene usato un numero di record non compreso tra 1 e 32767.
Can't continue	17	(Il programma non può proseguire) Si tenta di far riprendere l'esecuzione di un programma che – o è stato interrotto per un errore – o è stato modificato dopo l'interruzione – o non esiste
Disk full	61	(Disco pieno) Tutto lo spazio su disco è già occupato.
Disk I/O error	57	(Errore di accesso al disco) Questo messaggio viene visualizzato quando si richiede un'operazione di input/output su un disco ma il sistema non riesce ad accedervi.
Divisione by zero	11	(Divisione per zero)
Field overflow	50	(Campo troppo esteso) Si cerca di allocare in un buffer, mediante un FIELD, un numero di bytes inferiore a quello previsto in fase di apertura del file.
File already exist	58	(File già esistente) Si vuole assegnare ad un file un nome già utilizzato per un altro file.
File already open	55	(File già aperto) Si è cercato di aprire in modo sequenziale OUTPUT un file già aperto oppure di cancellare con un KILL un file ancora aperto.
File not found	53	(File non trovato) Si è fatto riferimento ad un file, inesistente in un LOAD, od in un KILL, oppure in un OPEN.

For without next	25	(FOR senza NEXT) Non è stato previsto il NEXT di chiusura per un ciclo FOR.
Illegal direct	12	(Uso improprio del 'direct mode') L'istruzione battuta non è eseguibile in 'direct mode'.
Illegal function call	5	(Errato riferimento ad una funzione) Viene usato un argomento non accettabile per la funzione. Es.: — un valore negativo per SQR — una lunghezza non compresa tra 0 e 255 per le funzioni LEFT$, MID$, RIGHT$.
Input past end	62	(Superamento della fine del file) Un'istruzione INPUT# viene richiesta quando non vi sono più elementi da leggere (o su un file vuoto). Per evitare questo tipo di errori si può utilizzare la funzione EOF.
Internal error	51	(Errore interno) Si è generato un errore di sistema.
Line buffer overflow	23	(Superamento della capacità del buffer) Si cerca di trasmettere un'istruzione formata da più di 255 caratteri.
Missing operand	22	(Operando mancante) Un'espressione contiene un operatore senza il relativo operando.
Next without for	1	(NEXT senza FOR) Viene incontrato un NEXT che non fa riferimento ad alcun FOR.
No resume	19	(Mancanza di RESUME) La routine per il trattamento dell'errore non contiene l'istruzione RESUME.

Out of data	4	(Dati finiti) Si tenta di eseguire un READ senza che in DATA vi siano ancora dei dati da leggere. – o si è dimenticato un DATA – o si è dimenticato un RESTORE
Out of memory	7	(Troppa memoria richiesta) Non vi è più spazio disponibile in memoria. Per proseguire bisogna sopprimere una parte del programma oppure sopprimere qualche vettore. (ERASE)
Out of string space	14	(Richiesta troppo elevata di spazi per le stringhe) Lo spazio assegnato alle stringhe non è più sufficiente (cfr CLEAR).
Overflow	6	(Superamento del valore limite) Si cerca di assegnare ad una variabile intera un valore non compreso tra –32768 e +32767.
Ridimensioned array	10	(Vettore ridimensionato) Un vettore viene dimensionato una seconda volta oppure si cerca di dimensionare un vettore già utilizzato in precedenza e dimensionato automaticamente a 10 dal sistema (per esempio in seguito ad un'istruzione del tipo: A(4)=X).
Resume without error	20	(RESUME senza errore) Viene incontrata l'istruzione RESUME senza che si sia riscontrato alcun errore.
Return without gosub	3	(RETURN senza GOSUB) – Si è entrati in un sottoprogramma con un GOTO anzichè con un GOSUB oppure – Si è entrati in un sottoprogramma per errore, avendo dimenticato nel programma principale un END o uno STOP (se il programma principale è stato scritto prima dei sottoprogrammi).

String formula too complex	16	(Espressione alfanumerica troppo complessa) L'espressione usata è troppo lunga oppure è troppo complessa. Occorre scomporla in espressioni più semplici.
String too long	15	(Stringa troppo lunga) Si vuole utilizzare una stringa con più di 255 caratteri.
Subscript out of range	9	(Indice non accettabile) Si fa riferimento ad un elemento di un vettore con un valore dell'indice che supera la dimensione del vettore. Questo errore è comune quando vengono utilizzati dei vettori senza dichiarare esplicitamente la dimensione, in quanto il sistema operativo li dimensiona automaticamente a 10.
Syntax error	2	(Errore di sintassi) L'istruzione contiene un errore di sintassi: — parametri mancanti — punteggiatura scorretta — istruzione sconosciuta — etc.
Type mismatch	13	(Incongruenza di tipo) Un valore numerico viene assegnato ad una variabile alfanumerica o viceversa.
Undefined line	8	(Linea inesistente) Si fa riferimento ad un numero di linea che non corrisponde ad alcuna istruzione.
Undefined user function	18	(Funzione non definita) Viene chiamata una funzione USR senza che sia stata prima definita.
Unprintable error	21	(Non vi sono messaggi per questo tipo di errore).

APPENDICE III

TABELLA DEI CODICI ASCII

0	@c		32	SPACE	64	@	96	`
1	Ac		33	!	65	A	97	a
2	Bc		34	"	66	B	98	b
3	Cc		35	#	67	C	99	c
4	Dc		36	$	68	D	100	d
5	Ec		37	%	69	E	101	e
6	Fc		38	&	70	F	102	f
7	Gc	(BEL)	39	'	71	G	103	g
8	Hc	(BS)	40	(	72	H	104	h
9	Ic	(HT)	41	)	73	I	105	i
10	Jc	(LF)	42	*	74	J	106	j
11	Kc		43	+	75	K	107	k
12	Lc		44	,	76	L	108	l
13	Mc	(CR)	45	-	77	M	109	m
14	Nc		46	.	78	N	110	n
15	Oc		47	/	79	O	111	o
16	Pc		48	0	80	P	112	p
17	Qc		49	1	81	Q	113	q
18	Rc		50	2	82	R	114	r
19	Sc		51	3	83	S	115	s
20	Tc		52	4	84	T	116	t
21	Uc		53	5	85	U	117	u
22	Vc		54	6	86	V	118	v
23	Wc		55	7	87	W	119	w
24	Xc		56	8	88	X	120	x
25	Yc		57	9	89	Y	121	y
26	Zc		58	:	90	Z	122	z
27	[c	(ESC)	59	;	91	[	123	{
28	\c		60	<	92	\	124	|
29	]c		61	=	93	]	125	}
30	^c		62	>	94	^	126	~
31	_c		63	?	95	_	127	DEL

APPENDICE IV

ELENCO DEI PROGRAMMI

NOTA: Nei programmi, a volte, sono state tralasciate alcune tra le routine più semplici, lasciando al lettore, quale esercizio, lo sviluppo delle stesse.

L. 11.000 Cod. 515 H ISBN 88-7056-117-8

Il libro si rivolge in modo particolare a chi già conosce il Basic e desidera poter realizzare programmi che prevedano l'uso di file residenti su disco. Per coloro che invece non hanno mai avuto occasione di utilizzare tale linguaggio una sezione a parte ne richiama le caratteristiche fondamentali. Dopo aver preso in esame, utilizzando numerosi esempi pratici, le particolarità della versione Basic scelta per le sue caratteristiche di generalità, (Microsoft, CP/M compatibile) si passa alla descrizione delle istruzioni necessarie ad una corretta gestione dei file su disco, sia ad accesso diretto che sequenziale. Anche in questo caso sono stati riportati numerosi esempi illustrati, riprodotti direttamente dai tabulati originali, al fine di evitare al massimo eventuali errori di trascrizione.
Una terza parte del libro è infine interamente dedicata alla esposizione dei metodi pratici per l'uso dei file ad accesso diretto e dei data base.
Il libro è concluso da due appendici tecniche, una sui messaggi d'errore e l'altra sui codici ASCII.

GRUPPO
EDITORIALE
JACKSON
Jacques Boisgontier
IL BASIC E LA GESTIONE DEI FILE
60